Semantics and Ambiguity of Stochastic RNA Family Models

Robert Giegerich and Christian Höner zu Siederdissen

Abstract

Stochastic models such as hidden Markov models or stochastic context free grammars can fail to return the correct, maximum likelihood solution in the case of semantic ambiguity. This problem arises when the algorithm implementing the model inspects the same solution in different guises. It is a difficult problem in the sense that proving semantic non-ambiguity has been shown to be algorithmically undecidable, while compensating for it (by coalescing scores of equivalent solutions) has been shown to be NP-hard. For stochastic context free grammars modeling RNA secondary structure, it has been shown that the distortion of results can be quite severe. Much less is known about the case when stochastic context free grammars model the matching of a query sequence to an implicit consensus structure for an RNA family.

We find that three different, meaningful semantics can be associated with the matching of a query against the model – a structural, an alignment, and a trace semantics. Rfam models correctly implement the alignment semantics, and are ambiguous with respect to the other two semantics, which are more abstract. We show how provably correct models can be generated for the trace semantics. For approaches where such a proof is not possible, we present an automated pipeline to check *post factum* for ambiguity of the generated models.

We propose that both the structure and the trace semantics are worth-while concepts for further study, possibly better suited to capture remotely related family members.

Index Terms

RNA secondary structure, RNA family models, covariance models, semantic ambiguity.

I. Introduction

A. Background: Semantics and ambiguity in stochastic modeling

Stochastic models: Stochastic models are powerful and widely used techniques in computational biology. In this article, we study covariance models implemented by stochastic context free grammars (SCFGs), which include hidden Markov models (HMMs) as a subclass. Let us start our discussion with this simpler model.

An important application of HMMs in biosequence analysis is the modeling of protein families. There, aligned protein sequences are processed into family models implemented as HMMs using the HMMer package [7] and stored in the Pfam data base [2]. Running a query sequence against a model returns a score that indicates the likelihood that the query belongs to the sequence family. Scanning a long sequence with the model reveals those regions that most likely share an evolutionary relationship with the model family.

An application of similar importance is the modeling of structural RNA families. Models are generated with the tool *Infernal* [8], [15] and accessed via the Rfam data base [9]. Here, an SCFG implements a covariance model of RNA sequences that share a consensus secondary structure. A “parse” of a query sequence with the model grammar shows how well it matches the family sequences, accounting for sequence as well as structure conservation.

HMMs and SCFGs use quite a different nomenclature. Nevertheless, mathematically, HMMs are a subclass of SCFGs – those cases where the context-free grammar underlying the SCFG belongs to the

Robert Giegerich is with the Center of Biotechnology and the Faculty of Technology at Bielefeld University, D-33615 Bielefeld, Germany; robert@techfak.uni-bielefeld.de

Christian Höner zu Siederdissen is with the Institute for Theoretical Chemistry, University of Vienna, Währingerstraße 17, 1090 Vienna, Austria; choener@tbi.univie.ac.at

subclass of regular grammars. Where the SCFG literature [8], [18] uses the terminology of formal language theory, such as grammars and parses, the HMM literature prefers a terminology of transition rules and paths. The CYK algorithm, which returns the highest scoring parse according to a SCFG, is a generalization of the Viterbi algorithm, which returns the highest scoring transition path for an HMM. In this article, we will build on the established SCFG terminology, because it makes the theory more general and also because our immediate practical interest lies with covariance models as used in Rfam.

Modeling semantic ambiguity: The problem of semantic ambiguity has been recently addressed in a series of papers. Giegerich [10] pointed out the problem and suggested a suitable formalization: the parse trees constructed by a SCFG parser represent some real-world objects of interest, for example alternative secondary structures for an RNA sequence. If some of these parses actually represent the same object, we have a case of semantic ambiguity. By specifying an explicit mapping of parses to a canonical (unique) representation of our objects of interest, it may be possible to prove presence or absence of ambiguity. The use of a canonical representation appears to be a necessary extension to the standard framework of stochastic modeling, in order to deal with ambiguity in a systematic manner. It plays the role of associating a precise semantics to the parse trees (namely, the structures they represent), and coding this meaning *within* the model is the key to tackling it computationally. The term "semantic" ambiguity that we use in this article catches this fact, and discerns it from syntactic ambiguity as studied in formal language theory. In our case, syntactic ambiguity only means that a grammar can specify several *different* structures for a given sequence, which is a good thing rather than a problem in combinatorial optimization. Note that in textbooks covering SCFGs [1], [6], the pitfall of semantic ambiguity has not yet been paid attention to, and the most likely parse is taken for granted to indicate the most likely structure.

Ambiguity – does it really matter: Dowell and Eddy [5] approached the ambiguity issue from the pragmatic side and investigated whether it really matters in practice. They compiled a number of plausibility arguments, why one might hope that the most likely parse somehow points to the most likely structure, even in the presence of ambiguity. But then, they refuted such hopes: For two ambiguous grammars, they tested how often the most likely parse returned by the SCFG was different from the most likely structure. For one grammar (G1), the result was wrong for 20% of all tested sequences. For the other grammar (G2), which was a refinement of G1 for the sake of better parameter training, the result was wrong even for 98%. Dowell and Eddy provided a first empirical test for the presence of ambiguity, and continued studying parameter estimation for several alternative, non-ambiguous grammars.

Algorithmic undecidability of semantic ambiguity: The idea of ambiguity checking was further worked out by Reeder et al. [16]. They gave a proof that, in general, presence or absence of semantic ambiguity is formally undecidable. However, they contributed a series of further techniques for ambiguity checking, where the most powerful one involves translation of the SCFG into a context-free grammar generating the canonical representation introduced in [10]. Then, a semi-decision procedure such as a parser generator may be able to demonstrate presence or prove absence of ambiguity in many relevant cases. The simple unambiguous grammars studied in [5] were proved unambiguous in this mechanized fashion. Moreover, the rather sophisticated grammar designed by Voss et al. for probabilistic shape analysis [21] could also be proved non-ambiguous in a similar way. The study by Reeder et al. [16] also indicated some techniques of avoiding ambiguity. However, there are cases where the expressiveness of the model – the capability of adapting the parameters of the model to a training set – may suggest to prefer a semantically ambiguous grammar.

Algorithmic infeasibility of ambiguity compensation: Can we still obtain the desired result when the grammar is ambiguous? Such a case was studied in the HMM literature by Brejova et al. [4] under the name "path labeling problem". In HMM modeling, the model itself often is more refined than the final result. For example, the gene structure of a sequence can be indicated by a labeling of residues by E (exon) or I (intron) states. Yeast, for example, has two classes of introns, "short" and "long". The stochastic model, in order to capture the length distribution of introns, requires several states to model intronic residues. Therefore, several transition paths through the model may differ in their points of transition between intronic states, while they lead to the same path labeling and hence, indicate the same

gene structure. Here, the path labeling constitutes the canonical mapping: Paths are equivalent when they have the same labeling, and the HMM is semantically ambiguous when this happens. Brejova et al. then studied what we call ambiguity compensation: can the algorithm be modified such that all scores of paths with the same labeling accumulate? Their main result was that, in general, this problem is NP-hard, and hence, computationally infeasible. This does not rule out that ambiguity compensation may be practical in restricted cases, but in general, we are better advised to avoid semantic ambiguity altogether.

Unresolved questions: There are three questions that were not addressed: (1) Dowell and Eddy studied semantic ambiguity in principle, but worked with rather small example grammars. Grammars such as those underlying Rfam are much larger, and they do not simply assign a structure to an RNA sequence, but they also relate it to the family model. It is unclear how the semantics of a model should be defined. (2) While methods for ambiguity checking in formal language theory have been advanced recently [3], the step from a large, tool-generated SCFG to the context-free grammar suitable for ambiguity checking is still left open. (3) Are the models used in practice actually semantically unambiguous, and if so, based on which semantics? These are the questions we will address.

B. Contributions of this article

This article provides a theoretical and a software-technical contribution, and their application to Rfam models.

On the theory side, we formally define three alternative semantics for covariance models for RNA families – a *structure*, a *trace*, and an *alignment semantics*. All three of them have a well-defined biological meaning, which is interesting to implement. Whether or not a particular grammar is in fact semantically ambiguous depends, of course, on the chosen semantics. We show how provably non-ambiguous models with respect to the trace semantics can be constructed.

On the technical side, we provide an automated pipeline that accepts a grammar G, a canonical representation mapping (written in a particular style), and produces a grammar $\widehat{G}$ which is *syntactically* ambiguous if and only if G is *semantically* ambiguous. Connecting this pipeline to a (syntactic) ambiguity checker for context-free grammars, this automates *semantic* ambiguity checking as far as its intrinsic undecidability allows for it.

In the application, we apply our formalism to Rfam models. We find that Rfam models faithfully implement the alignment semantics, although their description in the literature at one point suggests a structure semantics. With respect to the structure and the trace semantics, they are ambiguous. In the conclusion, we argue that both the structure and the trace semantics are worth further study, because they are more abstract and may be better suited to capture remotely related family members.

The article is organized as follows: In Section II we review what is known about semantics and ambiguity of simple SCFGs as used for structure prediction, about ambiguity checking, and ambiguity compensation. In Section III we turn to family model grammars and find that there are three alternative ways to define their semantics. In Section IV we describe precisely a new algorithm of model generation for the trace semantics and prove its correctness (i.e. non-ambiguity of the generated models). In Section V we describe a software for upward compilation and ambiguity checking of Rfam models. This pipeline is applied in Section VI. We conclude with a discussion of open research questions which arise from our findings.

II. A summary of semantic ambiguity theory

In this section, we review known results on the problem of semantic ambiguity. The only new contribution in this section is that the method for ambiguity checking suggested in [16] has now been automated. Along with this review, we introduce the concepts and the formalism to be further developed subsequently.

A. SCFGs and their semantic ambiguity

Context-free grammars: Given an alphabet $\mathcal{A}$ of symbols, $\mathcal{A}^*$ denotes the set of all strings of symbols from $\mathcal{A}$, including the empty string ϵ. A *context-free grammar* G is a formal system that generates a

$$G1: \quad S \to \varepsilon \,|\, aS \,|\, Sa \,|\, aSb \,|\, SS \qquad G5: \quad S \to \varepsilon \,|\, aS \,|\, aSbS$$

Fig. 1. Grammars G1 and G5 taken from [5]. S is the axiom and only nonterminal symbol in either grammar. a and b denote arbitrary bases out of $\{a, c, g, u\}$, as SCFGs allow non-standard base pairs (albeit with low probability). Hence, a rule like $S \to aSb$ is a shorthand for 16 different rules.

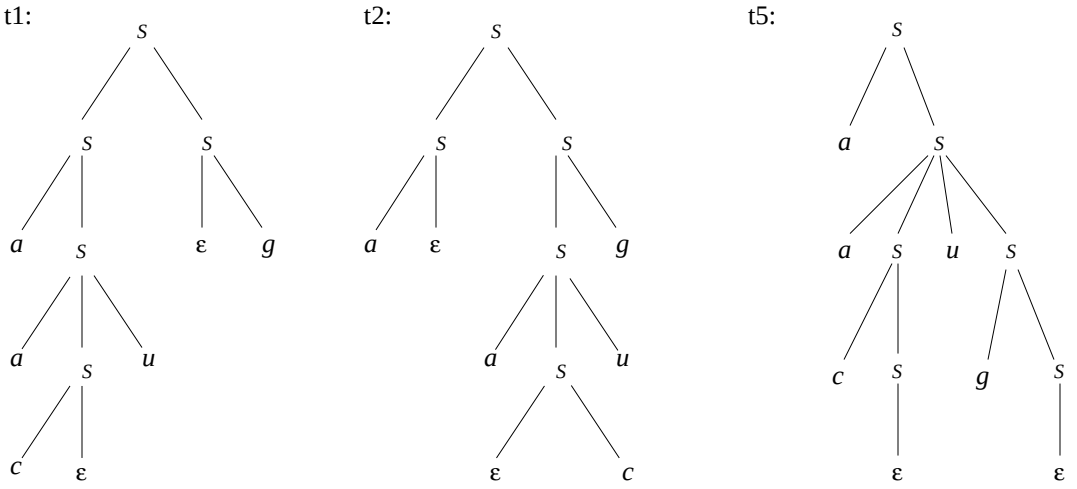

Fig. 2. Three derivation trees for the sequence `aacug`. $t1$ and $t2$ are derived with $G1$, $t5$ is derived with $G5$.

language of strings over $\mathcal{A}$. It uses a set V of *nonterminal symbols*, one of which is designated as the *axiom*. Its *derivation rules (productions)* have the form $X \to \alpha$, where $X \in V$ and $\alpha \in (V \cup \mathcal{A})^*$. A derivation of a terminal string $w \in \mathcal{A}^*$ starts from the axiom symbol, and in each step, replaces one of the nonterminal symbols in the emerging string according to one of the productions: $xXy \to x\alpha y$ may be a chosen transition when $X \to \alpha$ is a production of G. Such a derivation can be represented uniquely in the form of a tree, and by reversing the angle of view (from generating a string from the axiom to reducing a given string towards the axiom), this tree is also called a parse tree. Two grammars are shown in Fig. 1 , and three such parse trees are shown in Fig. 2. A grammar is *(syntactically) ambiguous* if there is a string that has at least two different parse trees. It is a classical result of formal language theory [12] that syntactic ambiguity of context-free grammars is formally undecidable. This means, there is no algorithm that can decide presence or absence of ambiguity *for all* context-free grammars. However, there are semi-decision procedures that return either YES, NO or MAYBE, which have proved quite powerful in practice [3].

Stochastic CFGs: A *stochastic* context-free grammar augments each production rule with a transition probability, such that the probabilities assigned with alternative rules for the same nonterminal symbol sum up to 1. For rules which simply generate a terminal symbol, the associated probability is called emission probability. We do not distinguish these two types of probabilities here. In a derivation, the probabilities of all applied rules multiply. In such a way, a parse tree t of string x assigns a probability $P(t, x)$ with x. The CYK algorithm, given x, computes the parse $t_{opt}(x) = argmax_t\{P(t, x) \mid t \text{ parse for } x\}$.

SCFG semantics: When modeling RNA structure, the *semantics* $\mathcal{S}_{\mathcal{SCFG}}$ of an SCFG G is defined as follows: Each parse tree t according to G associates an RNA secondary structure $\mathcal{S}_{\mathcal{SCFG}}(t)$ with sequence x: terminal symbols (denoting RNA bases) produced in the same step with productions like $S \to aSb$ are considered base paired, while all other ones are considered unpaired. Denoting structures in the familiar dot-bracket notation, where a dot denotes an unpaired base, and matching brackets denote paired bases, we observe $\mathcal{S}_{\mathcal{SCFG}}(t1) = \mathcal{S}_{\mathcal{SCFG}}(t2) = \mathcal{S}_{\mathcal{SCFG}}(t5) =$ `".(.)."`.

When there exist $t \neq t'$ but $\mathcal{S}_{\mathcal{SCFG}}(t) = \mathcal{S}_{\mathcal{SCFG}}(t')$ for grammar G, we say that G is *semantically ambiguous*. This occurs with the trees $t1$ and $t2$ for grammar $G1$ in Fig.2. There are no such trees with $G5$. Hence, $G1$ is semantically ambiguous, while $G5$ is an example of a non-ambiguous grammar.

With a semantically unambiguous grammar, the most likely parse also means the most likely structure for x – this is exactly what we hope to find. If the grammar is semantically ambiguous, the most likely structure s_{opt} may have several parses such that $s_{opt} = \mathcal{S}_{\mathcal{SCFG}}(t_1) = \mathcal{S}_{\mathcal{SCFG}}(t_2) = \ldots$, with probabilities $p(t_1, x), p(t_2, x), \ldots$, and $P(s_{opt}) = \sum_i p(t_i, x)$. In this situation, it is not guaranteed that one of the parses t_i has maximal probability, and some unrelated parse (indicating a different structure), will be returned by the CYK algorithm. For the grammars $G1$ and $G2$ studied in [5][1] , this happens in 20% resp. 98% of all test cases.

Many simple grammars can be specified for RNA structure that are not semantically ambiguous. Different (non-ambiguous) grammars for the same problem have different characteristics with respect to the probability distributions they define. For example, grammar $G5$, attributed to Ivo Hofacker in [5], is arguably the smallest grammar for the purpose. It has only 21 parameters and showed "abysmal" modeling performance in [5].

B. Embedding SCFGs in a more general framework

In order to deal with ambiguity checking and compensation, both in theory and practice, we embed SCFGs in the more general framework of algebraic dynamic programming (ADP) [11]. This will allow us to replace the probabilistic scoring scheme "hardwired" in the SCFG concept by other evaluation schemes, or use several such schemes in combination. In our application, we will in fact generate equivalent ADP code from Rfam models, to be used for a variety of different purposes aside from stochastic scoring.

Algebraic dynamic programming: ADP is a declarative method to design and implement dynamic programming algorithms over sequence data. ADP and stochastic modeling tools serve complementary purposes (while both rely on the same type of dynamic programming algorithms for their implementation). ADP is designed to give the author of a DP algorithm maximal convenience – high level of abstraction, re-usable components, and compilation into efficient target code. Any type of combinatorial optimization over sequences is possible, provided that Bellman's Principle of Optimality holds. Grammars in ADP are produced by a human designer and are typically small – at least compared to grammars derived from data by stochastic modeling tools. These, in turn, come with a hard-wired scoring scheme for maximizing probability or log-odds scores, and the capability to train the parameters via expectation maximization. Many of the grammars constructed by automatic modeling tools such as *Infernal* have probably never been inspected by a human eye.

The ADP formalism starts from a *signature*, which is a supply of function symbols[2]. One of these, named h by convention, designates the objective function, to be used in subsequent analyses. The other ones are placeholders for scoring functions.

For example, these are the signatures we will use with $G1$ and $G5$:

$G1$		G5	
$openl : \mathcal{A} \times V \to V$	$openr : V \times \mathcal{A} \to V$	$open : \mathcal{A} \times V \to V$	
$pair : \mathcal{A} \times V \times \mathcal{A} \to V$	$split : V \times V \to V$	$pair : \mathcal{A} \times V \times \mathcal{A} \times V \to V$	
$nil : V$	$h : [V] \to [V]$	$nil : V$	$h : [V] \to [V]$

Here, $\mathcal{A}$ denotes the underlying sequence alphabet, V an arbitrary value domain, and $[V]$ a list of values.

Grammars in ADP are tree grammars. A *tree grammar* is analogous to a context free grammar, except that the righthand side in $X \to \alpha$ now is a tree, built from the function symbols of the signature (other than h) at inner nodes, and nonterminal symbols as well as terminal symbols residing at the leaves of the

[1]Dowell and Eddy use the term "structural ambiguity" rather than "semantic ambiguity". This is consistent with our terminology, because for simple SCFGs, a structural semantics is the only one that has been considered so far. When we will turn to family models, there will be different semantics which can be employed. Again, there will be a structural semantics, but it is not the one implemented in today's modeling approaches.

[2]Java programmers may think of it as an interface

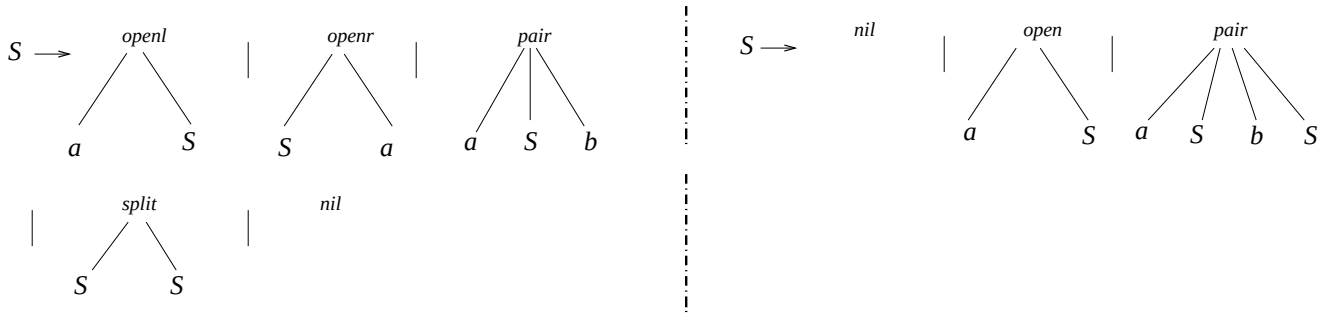

Fig. 3. Tree grammar versions of string grammars G1 (left) and G5 (right).

tree. Occasionally, we have a nullary function symbol, which also marks a leaf. Figure 3 shows the tree grammar versions of G1 and G5.

The derivation with a tree grammar works as with CFGs, except that now it produces a tree. It may derive the same tree in different ways (syntactic ambiguity of tree grammars), but this is easily avoided, and besides, syntactic ambiguity is decidable for this class of *tree* grammars. Therefore, we can assume that each tree has a unique derivation (or tree-parsetree). Each derived tree contains, as the string of its leaf symbols, some sequence $w \in A^*$. These trees represent the candidates in the combinatorial search space associated with sequence w, and in order to avoid the use of "tree" in too many connotations, we will henceforth refer to them as *candidates*.

The introduction of a tree grammar, based on a signature of functions, seems like a minor, artificial change of formalism, but has a profound impact: it decouples the candidates which we analyze from the grammar which generates them. There can be more functions in the signature than there are productions in the grammar, but normally, there are less. Different grammars over the same signature can be used to derive the same set of candidates. Candidates only reflect their signature – they bear no resemblance of the derivation and the grammar which generated them. Our candidates $t1, t2$ and $t5$ as derived by the tree grammars are shown in Figure 4.

The function symbols that constitute the inner nodes of the candidate can be used to associate a variety of meanings with each candidate. This is done by specifying an *evaluation algebra* – i.e. a data domain and a set of functions (which compute on this domain), one for each function symbol in the signature[3], including h. Whatever evaluation we define will be computed by a generic CYK-like algorithm. We do not worry about implementation issues here, and denote the analysis of input sequence x with grammar G and evaluation algebra B as a function call $G(B, x)$.

SCFGs encoded in ADP: To run an ADP grammar as a SCFG, one simply provides an evaluation algebra which implements the function symbols in the signature by functions that compute probabilities.

Evaluation algebra PROB for G1:

$$
\begin{array}{lcl@{\qquad}lcl}
h & = & \text{maximum} & & & \\
pair(a,x,b) & = & p_{ab} * x & split(x,y) & = & p_{split} * x * y \\
openl(a,x) & = & p_a * x & nil() & = & p_{nil} \\
openr(x,a) & = & p_a * x & & &
\end{array}
$$

The probability scores $p_a, p_{ab}, p_{split}, p_{nil}$ are to be estimated from the data.

Evaluation algebra PROB for G5:

$$
\begin{array}{lcl}
h & = & \text{maximum} \\
pair(a,x,b,y) & = & p_{ab} * x * y \\
open(a,x) & = & p_a * x \\
nil() & = & p_{nil}
\end{array}
$$

[3]Java programmers may think of implementing the "interface", but – please – with pure mathematical functions without side effects.

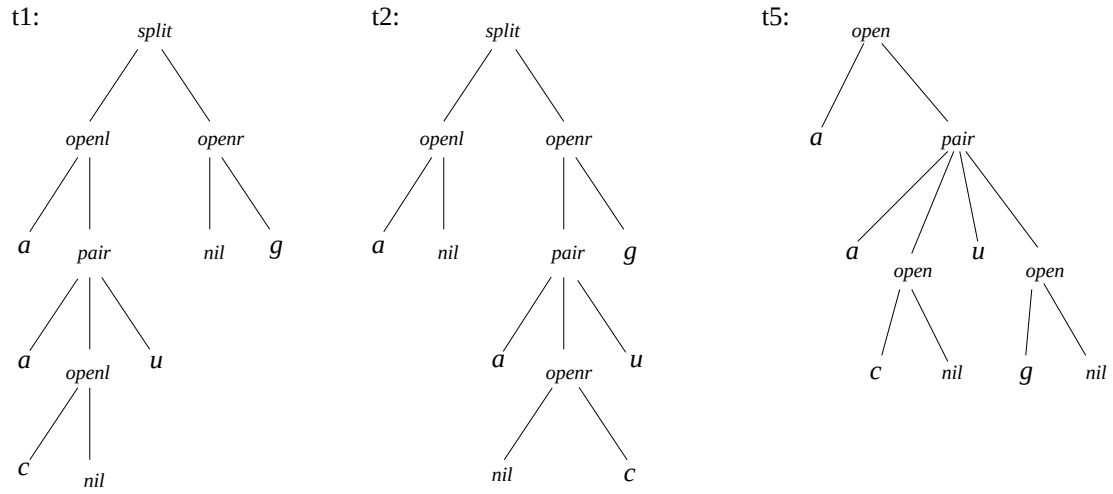

Fig. 4. Candidates $t1, t2$ and $t5$, as derived by their tree grammars $G1$ and $G5$.

Evaluating a candidate in this interpretation yields its probability score, akin to what is achieved by a SCFG, if the candidate was a parse tree. This is how we express the mathematical equivalent of an SCFG in ADP. The advantage is: once we have the grammar in ADP form, we can use it for other purposes besides stochastic scoring.

Encoding the canonical mapping: We use a second evaluation algebra to encode the canonical mapping of candidates to their "meanings". Let us call it CAN.

Evaluation algebra CAN for G1:

$$\begin{array}{lcllcl} h & = & id & & & \\ pair(a,x,b) & = & \texttt{"("} + x + \texttt{")"} & split(x,y) & = & x + y \\ openl(a,x) & = & \texttt{"}\cdot\texttt{"} + x & nil() & = & \texttt{""} \\ openr(x,a) & = & x + \texttt{"}\cdot\texttt{"} & & & \end{array}$$

In algebra CAN, we define the functions such that they compute, from the candidate, the dot-bracket representation of its associated structure. In other words, CAN implements the semantics $\mathcal{S}_{\mathcal{SCFG}}$ for G1. Operator $+$ here denotes string concatenation, and id denotes the identity function.

Evaluating the G1-candidates $t1$ and $t2$ in the algebras PROB and CAN, we obtain

PROB(t1)	$= split(openl(a, pair(a, openl(c, nil), u), openr(nil, g))$	$= p_{split} \cdot p_{nil} \cdot p_{au} \cdot p_a \cdot p_c \cdot p_g$
PROB(t2)	$= split(openl(a, nil), openr(pair(a, openr(nil, c), u), g))$	$= p_{split} \cdot p_{nil} \cdot p_{au} \cdot p_a \cdot p_c \cdot p_g$
CAN(t1)	$= split(openl(a, pair(a, openl(c, nil), u), openr(nil, g))$	$=$ `".(.)."`
CAN(t2)	$= split(openl(a, nil), openr(pair(a, openr(nil, c), u), g))$	$=$ `".(.)."`

In this way, the structure – the meaning of our candidates, which decides about ambiguity – now becomes part of our operational machinery. We can call $G1(CAN, \texttt{"aacug"})$, and multiple occurences of `".(.)."` in the output witness the semantic ambiguity of $G1$. We leave it to the reader to define an analogous algebras PROB and CAN for the signature of G5. A powerful feature of the ADP approach is the use of algebra products (see [20] for the precise definition). For example, calling $G5(PROB{*}CAN, x)$ will give us *all* the structures for x that achieve the maximum probability score. Since the grammar G5 is semantically non-ambiguous, there may still be several candidates achieving maximal probability, but they must all produce different structures as indicated by CAN. When the grammar is ambiguous (like G1), neither of the optimal candidates may indicate the most likely structure, as explained in Section II-A. $G1(PROB * CAN, x)$ returns the optimal candidates together with their associated structures, possibly delivering duplicates, but we cannot be sure if any of them denotes the most likely structure.

$$\widehat{G1}: \qquad S \to (S) \mid .S \mid S. \mid SS \mid \varepsilon$$

Fig. 5. Grammar $\widehat{G1}$ derived from G1

C. Automated checking of semantic ambiguity

We now introduce a systematic and automated approach to ambiguity checking. Consider a tree grammar and a canonical mapping algebra which maps candidates to strings over some alphabet $\widehat{A}$. In this setting, one can substitute the string composing functions of the algebra into the righthand sides of the tree productions. By partial evaluation, we eliminate the trees, and righthand sides become strings over $V \cup \widehat{A}$. Starting from the tree grammar G1, we rewrite its rules into those of grammar $\widehat{G1}$, shown in Fig. 5.

Note that the first rule in $\widehat{G1}$ is derived from 16 productions in G1, but since these are mutually exclusive due to their terminal symbols, only one corresponding rule is retained in $\widehat{G1}$.

In this way, from our tree grammar G we obtain a context-free (string) grammar $\widehat{G}$ with the following property:

Theorem 1 The tree grammar G is semantically ambiguous if and only if the string grammar $\widehat{G}$ is syntactically ambiguous.

The proof of this theorem was given in [16]. At that time, the grammar $\widehat{G}$ was handwritten – the new aspect here is that it is now produced automatically from G and the canonical mapping algebra. This is further described in Section V, where we present the pipeline *cm2adp* for upward compilation of *Infernal*-generated models into the ADP framework. Taking these constituents together –

1) the automated re-coding of an SCFG in ADP as a tree grammar G,
2) the specification of a unique string representation as canonical mapping algebra CAN,
3) the automated derivation of a string grammar $\widehat{G}$ from G and CAN,

we are now in a state where we can take a SCFG and submit it to an automatic ambiguity checker.

The only step which is not automated is, of course, the specification of the canonical mapping CAN. Naturally, we must say at one point what the meaning of our candidates really is. However, for grammars coming from the same modeling domain, this must be done only once, as the canonical mapping is the same for all grammars. In this sense, the ambiguity checking pipeline is completely automated now.

D. Ambiguity compensation

The canonical mapping defines (as its reverse image) a semantic equivalence relation on the evaluated candidates. Ambiguity compensation means that all scores within the same equivalence class should be accumulated, rather than maximized over. Let us assume for the moment that we know how to accumulate these scores[4]. We obtain an accumulating algebra $PROB_{acc}$ from $PROB$ by replacing the (maximizing) objective function h by the suitable accumulating function h_{acc}. By calling $G1(CAN * PROB_{acc}, x)$, we correctly compute the probabilities, accumulated over the equivalence classes modulo CAN. So, mathematically, ambiguity compensation is not a problem, and no additional programming effort is required except for the coding of h_{acc}.

However, we will experience an exponential slowdown of our program, consistent with the intractability result of [4]. The asymptotic efficiency of the algorithm is affected by the number of equivalence classes modulo CAN, which must be computed in total – and their number is, in general, exponential in the length of the input sequence. Such an approach is feasible for moderate length RNAs when equivalence classes are defined via shape abstractions [21], but when CAN simply denotes feasible structures of the input sequence, one cannot get very far by this (otherwise quite elegant) method.

[4]For example, log-probabilities must be re-converted into probabilities in order to be added, which may cause numerical problems.

E. Ambiguity in sequence comparison: alignments versus traces

The phenomenon of semantic ambiguity is not peculiar to SCFGs. It arises with HMMs, and in fact, already with simple, pairwise sequence alignments. As with SCFGs, it depends on the meaning we associate with alignments. Seen as a syntactic object, each alignment of two sequences x and y stands for itself and is distinct from all others. But sequence alignments are often interpreted as a reconstruction of evolutionary history, where both sequences have developed from a common ancestor. Matched residues in the alignment (bases for DNA, amino acids for protein sequences) are considered preserved by evolution. Mismatches mean accepted point mutations. Gaps mean new residues that have been inserted in either x or y. (If we see the same process as evolving from x to y, "insertions" in x appear as deletions, which has no effect on the subsequent discussion.) If new sequence has been inserted in both x and y between two preserved residues, there is no particular ordering of these events. The alignment, however, offers two representations for the same fact: we may write both

```
x:   ACAGGGG---CAC          x:   ACA---GGGGCAC
y:   ACA----TTTCAC          y:   ACATTT----CAC,
```

denoting the same evolutionary history. Classical bioinformatics textbooks do not fail to point to this fact [19], [22]. Naturally, if this situation arises at k locations during the evolution of the sequences, this process has 2^k aligments representing it – significantly disturbing any stochastic model.

"Alignments" where only matches and mismatches are specified, and hence, adjacent deletions/insertions remain implicit, avoid this problem. They are called *traces* in [19], and we will adopt this naming later. An unambiguous notation for traces could be e.g.

```
x:   ACA[GGGG]CAC
y:   ACA[TTT] CAC
```

where the square brackets designate inserted sequences unordered with respect to each other. Another way to avoid ambiguity in alignments is presented later, when we return to this aspect in Section III-C.

III. Semantics of SCFG-based family models

In this section we turn our attention to SCFGs which describe RNA family models, called family model grammars for short. The previously developed SCFG terminology is not sufficient to understand their properties. We will extend it appropriately. In particular, we will find that there are three reasonable, alternative semantics for family model grammars.

A. From RNA folding SCFGs to family model grammars

There are three important differences between the SCFGs as we (and others) have used them as models for structures of individual RNA molecules, and their use in family modeling.

Family model grammars encode a consensus structure: Grammars like $G1$ or $G5$ are unrestricted RNA folding grammars. They will fold a sequence into all feasible secondary structures according to the rules of base pairing. This makes the grammars relatively small, having one rule for every structural feature considered by the scoring scheme, say a base pair or an unpaired base. The scoring scheme evaluates alternative parses and selects the result from the complete folding space of the query sequence.

This is different with grammars that model an RNA family with a particular consensus structure C. The consensus structure C is "hard-coded" in the grammar. To show a concrete consensus, we shall use star and angle brackets in place of dots and parenthesis, e.g. `"*<<*<*>>><*>*"`. This is only for clarity – there is no difference, in principle, between the consensus and ordinary structures. For every position where (say) a base pair is generated, the family model grammar has a special copy of the base pair generating production, with nonterminal symbols renamed. The general rule $S \rightarrow aSu$ becomes $S_i \rightarrow aS_{i+1}u$ for each position i where an $a{-}u$ base pair is in C. The transition parameter associated with this rule can be

trained to reflect the probability of an $a-u$ pair in this particular position. The type of grammar we have seen before, therefore, only serves as a prototype from which such position-specific rules are generated.

A family consensus structure of n residues will lead to a model grammar G_C with kn productions, where k is a small constant. Hence, while folding a query with approximate length n and grammar $G1$ would require $O(n^3)$ computing steps, matching the sequence to the family grammar G_C runs in $O(n^4)$ time, simply because the size of G_C is in $O(n)$.

Family model grammars restrict the folding space of the query: A parse of a sequence x in G_C indicates a structure for x, but this structure is no longer a free folding: it is always a homomorphic image of C, with some base pairings of C possibly missing, and some residues of C possibly deleted. Still, the paired residues may be assigned to the bases of x in different ways; therefore, the structures assigned to x by different parses may vary slightly. This restriction of the folding space to "lookalikes" of C is the second difference between single sequence folding and family modeling.

Family model grammars encode the alignment of a query to the consensus: The third, important difference is that G_C implicitly aligns x to C. For example, a base assigned an unpaired status in x may represent one of three situations: it may (i) be matched to an unpaired residue in C, (ii) be an inserted base relative to C, or (iii) be matched to a paired residue in C, but without having a pairing partner in x.

These three situations are explicitly distinguished in G_C, they are scored separately, and the CYK algorithm returns the parse with maximal score based on these considerations. To achieve this, the prototype grammar needs rules which take care of deletions, insertions, and different types of matches.

Together, these three differences are central to our issue of ambiguity, and we summarize them in the following

Fact *Let M be a covariance model implemented by an SCFG G_C, which implicitly encodes the consensus structure C. Then, parsing x with G_C finds an optimal alignment of x with C which implicitly designates a structure s_x for x. This structure s_x is restricted to one of many possible homomorphic images of C obtained by deleting residues and dropping base pairings from C. There are numerous other alignments which assign the same structure s_x to x, whose (smaller) likelihood contributions are not reflected by the optimal alignment.*

B. Prototype grammar and family model example

At this point the reader rightfully expects an example of a prototype grammar and a family model grammar generated from it. We show a prototype grammar derived from $G5$ and a toy family model grammar generated from it.

The prototype grammar G5M: We extend $G5$ to obtain a prototype grammar $G5M$ capable of describing query alignments to a model. $G5M$ extends $G5$ by rules modeling insertions, deletions and matches. Again, a and b stand for arbitrary bases.
Grammar $G5M$, the axiom is A.

$$\begin{array}{lcl} A & \rightarrow & a\ A \mid M \\ M & \rightarrow & \varepsilon \mid a\ A \mid M \mid \\ & & a\ A\ b\ A \mid a\ A\ M \mid M\ b\ A \mid M\ M \end{array}$$

From a purely syntactic point of view, this grammar appears weird, because the chain rule $M \rightarrow M$ and $M \rightarrow M\ M$ together with $M \rightarrow \varepsilon$ allow for unbounded derivations that produce ε. There is no string in the language of this grammar which has a unique derivation! Ignoring all rules except $\{M \rightarrow \varepsilon,\ M \rightarrow a\ A,\ M \rightarrow a\ A\ b\ A\}$ and mapping nonterminal symbols A and M to S, we are back at $G5$. The other rules provide for insertions and deletions between the query and the model. Specialization of $G5M$ to the consensus `"*<*>*"` will yield the family model grammar G_{Toy5}. Its context-free core is shown in Fig. 6 for shortness, but G_{Toy5} actually is a tree grammar using the same signature as $G5M$. Details of the generation algorithm are in Section IV.

To make our intentions explicit, we semantically enhance the grammars by adding an evaluation function interface.

$$
\begin{array}{ll}
A_1 & \rightarrow a\ A_1 \mid M_1 \\
M_1 & \rightarrow a\ A_2 \mid M_2 \\
A_2 & \rightarrow a\ A_2 \mid M_2 \\
M_2 & \rightarrow a\ A_3\ b\ A_5 \mid a\ A_3\ M_5 \mid\ M_3\ b\ A_5 \mid\ M_3\ M_5 \\
A_3 & \rightarrow a\ A_3 \mid M_3 \\
M_3 & \rightarrow a\ A_4 \mid M_4 \\
A_4 & \rightarrow a\ A_4 \mid M_4 \\
M_4 & \rightarrow \varepsilon \\
A_5 & \rightarrow a\ A_5 \mid M_5 \\
M_5 & \rightarrow a\ A_6 \mid M_6 \\
A_6 & \rightarrow a\ A_6 \mid M_6 \\
M_6 & \rightarrow \varepsilon
\end{array}
$$

Fig. 6. Family model grammar G_{Toy5} generated from $G5M$ for consensus $C =$ "*<*>*"

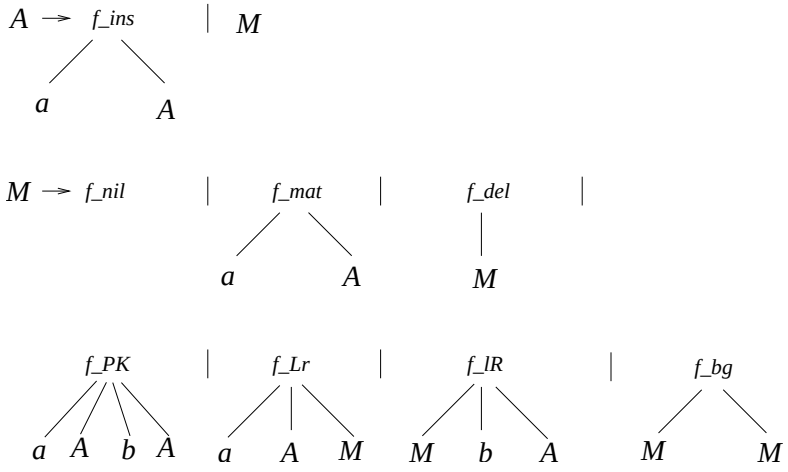

Fig. 7. Prototype grammar $G5M$ as a tree grammar. Functions f_{mat}, f_{ins} and f_{del} mark matches, insertions and deletions of unpaired residues. Functions f_{PK}, f_{Lr}, f_{lR}, and f_{bg} mark matches, partial, or total deletions of paired residues in the model.

Here is the signature:

$$
\begin{array}{llll}
f_{mat}: & \mathcal{A} \times V \rightarrow V & f_{PK}: & \mathcal{A} \times V \times \mathcal{A} \times V \rightarrow V \\
f_{ins}: & \mathcal{A} \times V \rightarrow V & f_{Lr}: & \mathcal{A} \times V \times V \rightarrow V \\
f_{del}: & V \rightarrow V & f_{lR}: & V \times \mathcal{A} \times V \rightarrow V \\
f_{nil}: & V & f_{bg}: & V \times V \rightarrow V \\
h: & [V] \rightarrow [V] & &
\end{array}
$$

Remember that $\mathcal{A}$ denotes the underlying alphabet. The tree grammar version of $G5M$ is shown in Fig. 7.

C. Three semantics for family model grammars

Matching a query x against a family model should return the maximum likelihood score of – what? There are three possibilities, which we will explicate in this section.

For the family models, derived from $G5M$, we can use the same signature as with $G5M$, except that the functions get, as an extra first argument, the position in the consensus with which they are associated. Hence, when specifying a semantics via an evaluation algebra for $G5M$, this implies the analog semantics for all generated models, as they solely consist of position-specialized rules from $G5M$.

The structure semantics: The obvious idea is to ask for the highest scoring structure assigned to x. This is in line with the semantics $\mathcal{S}_{\mathcal{SCFG}}$ introduced for SCFGs previously. Here is the canonical mapping algebra CAN_{struct}:

$$\begin{array}{llll} f_{mat}(a,s) & = \text{"."} + s & f_{PK}(a,s,b,t) & = \text{"("} + s + \text{")"} + t \\ f_{ins}(a,s) & = \text{"."} + s & f_{Lr}(a,s,t) & = \text{"."} + s + t \\ f_{del}(s) & = s & f_{lR}(s,b,t) & = s + \text{"."} + t \\ f_{nil} & = \text{""} & f_{bg}(s,t) & = s + t \\ h & = id & & \end{array}$$

Here again, s and t denote strings derived from substructures, and $+$ denotes string concatenation. a and b are the concrete residues read in the query. CAN_{struct} maps residues of the query x to their assigned paired or unpaired status, while residues from the consensus which are deleted (e.g. where f_{bg} applies) produce no contribution to the output. Hence, the meaning of any candidate evaluated with CAN_{struct} is simply a structure for x in dot-bracket notation.

The alignment semantics: With the alignment semantics, we want to obtain the maximum likelihood score of an alignment of the query to the consensus. This model is more refined than the structure semantics, as a given query structure can be aligned to the consensus in many different ways, and we seek the most likely of those. Let us now formalize this idea.

For capturing the alignment semantics, we must use a canonical representation that expresses not only the structure assigned to x, but also how it is aligned to the consensus structure C. Hence, it is an alignment of two strings, the consensus structure and the structure assigned to the query. Both, naturally, can be padded with gaps. The following are three different alignments of a query sequence to the same consensus:

```
(1)   **<<**--*>*>-      (2)      **<-<*----**>*>      (3)      **<<**-*>*>--
      ._(......__).               ____.(......)_.               __((....)_)..
```

Note that the upper line is always `"**<<***>*>"` when ignoring the gaps. This is because the consensus is hard-coded in the model grammar. In contrast, the query structure is `".(......)."` in alignments (1) and (2), and `"((....)).."` in alignment (3).

In defining the canonical mapping algebra CAN_{align} for the alignment semantics, we use functions that generate the alignment column-wise.[5]

Here is the canonical mapping CAN_{align}:

$$\begin{array}{llll} f_{mat}(a,s) & = \text{"}\overset{*}{.}\text{"} + s & f_{PK}(a,s,b,t) & = \text{"}\overset{<}{(}\text{"} + s + \text{"}\overset{>}{)}\text{"} + t \\ f_{ins}(a,s) & = \text{"}\overset{-}{.}\text{"} + s & f_{Lr}(a,s,t) & = \text{"}\overset{<}{.}\text{"} + s + \text{"}\underline{>}\text{"} + t \\ f_{del}(s) & = \text{"}\underline{*}\text{"} + s & f_{lR}(s,b,t) & = \text{"}\underline{<}\text{"} s + \text{"}\overset{>}{.}\text{"} + t \\ f_{nil} & = \text{""} & f_{bg}(s,t) & = \text{"}\underline{<}\text{"} + s + \text{"}\underline{>}\text{"} + t \\ h & = id & & \end{array}$$

The trace semantics: Our third semantic idea results from the fact the good old sequence alignments have an ambiguity problem of their own. After all, we are aligning a query sequence to the model. Recall our example from Section II-E of traces of evolutionary processes that are represented ambiguously by sequence alignments:

```
x:   ACAGGGG---CAC          x:   ACA---GGGGCAC
y:   ACA----TTTCAC          y:   ACATTT----CAC
```

This directly pertains to our problem at hand if you consider x as the consensus of (say) a loop region in the model (be it a profile HMM or an SCFG), and y as the loop of a corresponding hairpin in the

[5]In the implementation, unfortunately, we have to replace the nice two-letter columns by ASCII encodings.

query. If a stochastic model assigns a probability of 0.1 to each of the two alignments, the corresponding trace

```
x:   ACA[GGGG]CAC
y:   ACA[TTT] CAC
```

has probability 0.2 (at least). We have a case of semantic ambiguity, which must be taken care of even in stochastic sequence alignment. The Plan7 architecture of HMMer, for example, does this in a drastic way by requiring at least one intervening match when switching between deletions and insertions [17]. This simply disallows adjacent deletions and insertions altogether (but also rules out some plausible traces).

We will adopt a different route. It is easy to modify the alignment recurrences defining sequence alignment (the grammar in our terminology) such that only one of the possible arrangements of adjacent insertions and deletions is considered as a legal alignment [10]. With such canonization, each trace is uniquely represented by an alignment. The reduction is significant: For the two short sequences shown above, and under the affine gap model, there are 396,869,386 alignments, representing only 92,378 different traces[6]. Traces are considerably more abstract than alignments.

Let us return to our covariance models. Our family model grammars perform both folding and alignment, and hence, they are also affected by this source of ambiguity – at least if we intend that final score designates the most likely evolutionary process that relates the query to the model. The case even becomes more subtle. The following alignment (4) denotes the same trace as alignment (2):

```
(2) C:    **<-<*----**>*>        (4) C:    **-<<*----**>*>
    x:    ___.(......)_.             x:    __._(......)_.
```

What both alignments say is that a paired residue (at position 3) in the consensus C is deleted in x, while another base is inserted in x. As with plain sequence alignments, adjacent deletions and insertions are unrelated; their order is insignificant.

Hence, it makes sense to introduce a *trace semantics* for our family model grammars: we want to obtain the maximum likelihood score of a trace, which uniquely describes an evolutionary process of transforming the consensus into the query.

To capture this idea, we need to design another canonical algebra CAN_{trace}, which maps these two situations (2) and (4) above to the same, unique representation. Let us adopt the canonization rule that insertions must always precede adjacent deletions. By this rule, both alignments (2) and (4) are represented in the form of (4). The canonical mapping algebra CAN_{trace} is almost the same as CAN_{align}, except that deletions that appear to the left of an insertion are pushed to the right.

Algebra CAN_{trace}

$$
\begin{array}{lll}
f_{mat}(a,s) & = \text{"}\overset{*}{.}\text{"} + s & f_{PK}(a,s,b,t) = \text{"}\overset{<}{(}\text{"} + s + \text{"}\overset{>}{)}\text{"} + t \\
f_{ins}(a,s) & = \text{"}\overset{-}{.}\text{"} + s & f_{Lr}(a,s,t) = \text{"}\overset{<}{.}\text{"} + s + \text{"}\overset{>}{\_}\text{"} \triangleright t \\
f_{del}(s) & = \text{"}\overset{*}{\_}\text{"} \triangleright s & f_{lR}(s,b,t) = \text{"}\overset{<}{\_}\text{"} \triangleright s + \text{"}\overset{>}{.}\text{"} + t \\
f_{nil} & = \text{""} & f_{bg}(s,t) = \text{"}\overset{<}{\_}\text{"} \triangleright s + \text{"}\overset{>}{\_}\text{"} \triangleright t \\
h & = id &
\end{array}
$$

$$
\begin{array}{lll}
d \triangleright (a+s) & = \text{if } a = \text{"}\overset{-}{.}\text{"} & \text{then } a + (d \triangleright s) \quad \text{else } d + a + s \\
d \triangleright \varepsilon & = d &
\end{array}
$$

Wherever a deletion is issued, we have replaced simple string concatenation (+) by the operation $\triangleright$ which moves the deletion to the right over any leading insertions.

[6]Computed with the ADP versions of classical dynamic programming algorithms at http://bibiserv.techfak.uni-bielefeld.de/adp/adpapp.html

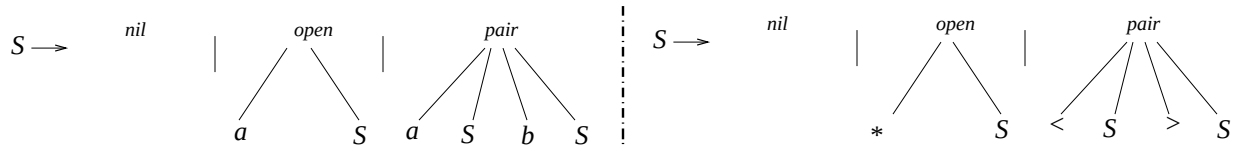

Fig. 8. Grammar $G5$ as a tree grammar for parsing sequences (left) and consensus structures (right).

The semantic hierarchy: Our three semantics form a proper hierarchy – many alignments correspond to the same trace, and many traces assign the same structure to the query. This also implies that a family model which faithfully (unambiguously) implements the alignment semantics is ambiguous with respect to the trace semantics, and one which faithfully implements the trace semantics is ambiguous with respect to the structure semantics, which is the most abstract of the three.

Which semantics to choose? When we are mainly interested in a structure prediction for the query, indicating why x may perform the same catalytic or regulatory function as the family members, then the structure semantics may be most appropriate. When we are interested in estimating the evolutionary closeness of the query to the family members, the trace semantics seems adequate. For the alignment semantics, at the moment we see no case where it should be preferred.

But – can we generate unambiguous family model grammars and efficiently compute either of the three semantics?

IV. Generating non-ambiguous family models for the trace semantics

In this section we show how family model grammars can be generated which are non-ambiguous with respect to the trace semantics. This will also provide a deeper insight on the meaning of the prototype grammar [7]. We proceed in the following steps: (1) We start from a non-ambiguous prototype grammar. (2) We show how, given a consensus structure C, a model grammar G_C is constructed which generates alignments in the canonical form (insert-before-delete), as required for the trace semantics. (3) We give a proof that for any C, G_C is non-ambiguous under the trace semantics.

Here, we use grammar $G5$, because it is the smallest non-ambiguous grammar. However, the generating technique and proof carries over to any non-ambiguous prototype grammar, which might be more attractive than G5 from the parameter training point of view.

The meaning of prototype grammars: Starting from $G5$, our prototype grammar is $G5M$. We still owe the reader the explanation why this grammar looks the way it does. The key point of $G5M$ is that it enforces the insert-before-delete convention. Only nonterminal symbol A allows for insertions. Whenever a nonterminal symbol stands in the left context of a deletion, an M rather than an A is used.

The real understanding of the prototype grammar comes from the observation that the prototype grammar $G5M$ is a grammar that allows to align a query to *all* possible models:

There is no *specific* model encoded in $G5M$. This is why the grammar can be so small. But each derivation with $G5M$ not only assigns a structure to the query, but also implicitly encodes a model, chosen by that derivation. This meaning of the prototype grammar can be made apparent by plugging the definitions of CAN_{trace} into the tree grammar $G5M$ and symbolically evaluating a bit. Doing so, the tree operators like f_{mat} or f_{PK} are replaced by string concatenations, and we obtain the string grammar $\widehat{G5M}$:

Grammar $\widehat{G5M}$; the axiom is A.

[7]The reader may find it helpful to inspect the actual implementation of the generator and run simple experiments. We therefore have provided the generator among our educational ADP pages at http://bibiserv.techfak.uni-bielefeld.de/adp/nilpairopen.html

$$
\begin{aligned}
A &\rightarrow \bar{\bullet}\ A \mid M \\
M &\rightarrow \varepsilon \mid \overset{*}{\bullet}\ A \mid \underline{\overset{*}{\ }}\ M \mid \\
&\quad \overset{<}{(}\ A\ \overset{>}{)}\ A \mid \overset{<}{\bullet}\ A\ \underline{\overset{>}{\ }}\ M \mid \underline{\overset{<}{\ }}\ M\ \overset{>}{\bullet}\ A \mid \underline{\overset{<}{\ }}\ M\ \underline{\overset{>}{\ }}\ M
\end{aligned}
$$

This grammar transformation is not totally trivial because of the use of $\triangleright$ in the definitions of CAN_{trace}. But from the grammar, we observe that canonical strings derived from M cannot start with insertions ($\bar{\bullet}$), while deletions ($\underline{\overset{*}{\ }}, \underline{\overset{<}{\ }}, \underline{\overset{>}{\ }}$) are only applied before M. Hence, this grammar guarantees that the if-clause in the definition of $\triangleright$ is never positive, and the recursive call to $\triangleright$ disappears. (Since the use of $\triangleright$ is the only difference between CAN_{trace} and CAN_{align}, transforming $G5M$ with CAN_{align} leads to the same string grammar $\widehat{G5M}$).

What does $\widehat{G5M}$ explain about $G5M$? Replaying any derivation of $G5M$ with the analog productions of $\widehat{G5M}$ produces the representation of a model-structure alignment. The top line displays the model "chosen" in this derivation, the bottom line displays the structure assigned to the query. Considering only the model string on the top line, we find that its is produced by productions analog to $G5$, and hence, any consensus structure is possible.

For example, running $\widehat{G5M}$ on input `"au"` produces an infinite number of model/query alignments. This is correct, since models of any length can be aligned to any sequence with a suitable number of deletions. Disabling for a moment the rules which delete unpaired model residues or both residues in a pair (i.e. the uses of f_{del} and f_{bg}), which are the sources of such infinity, the prototype grammar $\widehat{G5M}$ generates the following 23 alignments via the call $G5M(CAN_{trace}, \texttt{"au"})$:

"--" ".."	"**" ".."	"<>*" "._."	"<<>>" ".._"	"<><>" "_._."	"-*" ".."	"*<>" ".._"	"<><>" "._._"	"<<>>" "._._"	"<*>" "_.."
"-<>" ".._"	"*<>" "._."	"<><>" "._."	"<>-" "_.."	"<<>>" "_._."	"-<>" X "._."	"<>" "()"	"<->" X ".._"	"<>*" "_.."	"<<>>" "__.."
"*-" ".."	"<*>" ".._"	"<><>" "_._."							

Note that we see two alignments (labeled X) that satisfy the insert-before-delete convention, but not their counterparts with delete-before-insert, which is forbidden with the trace semantics. Let us summarize our observations about the role of the prototype grammar.

Fact *The prototype grammar describes, by virtue of its derivations, the alignment of a query to all possible consensi. Generating a specific family model grammar amounts to restricting the prototype grammar, such that all its derivations align the query to the same model consensus.*

In other words, in a family model grammar for consensus structure C, the "upper line" in a derivation always spells out C.

Generating model grammars from consensus structures: We now construct a generator which reads a consensus structure C such as "`***<<<******>>*>**`" and generates a grammar $G5M_C$ which implicitly encodes alignments of a query sequence x to C. With the ADP method at our disposal, we can use a variant of tree grammar $G5$ to parse C, obtained by substituting `*` for unpaired residues and `<` and `>` for paired ones (cf. Fig. 8 (right)). Since $G5$ is non-ambiguous, there will be only one tree t_C for C. We design an evaluation algebra $genCM$ which generates $G5M_C$ by evaluating t_C. For the sake of explanation, we will proceed in two steps: first we design an algebra $genCFG$ which generates $G5M_C$ as a context free grammar, to explain the logic of the algorithm. Then, we modify $genCFG$ to $genCM$ which generates a tree grammar, i.e. executable ADP code for the model.

$genCFG$ has to take care of two issues. (1) It must generate copies of the rules of $G5M$, specialized to the specific positions in C. Applying (say) rule $M \rightarrow aAbA$ when a and b are at paired positions i and j in C, respectively, will produce the specialized production $M_i \rightarrow aA_{i+1}bA_{j+1}$. (2) $genCFG$ must allow for insertions and deletions without introducing ambiguity. But this has already been taken care of in the design of $G5M$. As long as $genCFG$ only uses position-specialized copies of the rules from $G5M$, this property is inherited.

Evaluation algebra $genCFG$; the value domain is sets of context-free productions:

$$
\begin{aligned}
nil(\varepsilon_i) &= \{A_i \rightarrow a\ A_i \ \mid\ M_i,\ M_i \rightarrow \varepsilon\} && (1)\\
open(a_i, x) &= x \cup \{A_i \rightarrow a\ A_i \ \mid\ M_i\} && (2)\\
&\quad \cup \{M_i \rightarrow a\ A_{i+1} \ \mid\ M_{i+1}\} && (3)\\
pair(a_i, x, b_j, y) &= x \cup y \cup \{A_i \rightarrow a\ A_i \ \mid\ M_i\} && (4)\\
&\quad \cup\ M_i \rightarrow\ a\ A_{i+1}\ b\ A_{j+1}\} && (5)\\
&\quad \cup\ M_i \rightarrow\ a\ A_{i+1}\ M_{j+1}\} && (6)\\
&\quad \cup\ M_i \rightarrow\ M_{i+1}\ b\ A_{j+1}\} && (7)\\
&\quad \cup\ M_i \rightarrow\ M_{i+1}\ M_{j+1}\} && (8)
\end{aligned}
$$

Here, subscripts denote the position where a particular production is applied in the parse of C. In the output, these numbers create nonterminal symbols distinguished by subscripts. By default, the axiom of the generated grammars is A_1. Our reader may verify: computing $G5(genCFG, \texttt{"*<*>*"})$ yields the grammar $Toy5$ shown in Fig. 6.

Finally, to produce executable code, $genCM$ must generate a tree grammar rather than a string grammar, in order to integrate the scoring functions. The rules of the context-free grammar derived with $genCFG$ are now associated with scoring functions from the signature. As we cannot produce graphical output, a tree build from function symbol f and subtrees a, A, b, A is coded in the form `f <<< a~~~A~~~b~~~A`.

Evaluation algebra $genCM$; the value domain is sets of tree grammar productions written in ASCII:

```
nil(ε_i) = {A_i = f_ins <<< a ~~~ A_i ||| M_i}                          (9)
         ∪ {M_i = f_nil<<<empty}                                        (10)
open(a_i,x) = x ∪ {A_i = f_ins <<< a ~~~ A_i ||| M_i}                  (11)
         ∪ {M_i = f_mat <<< a ~~~ A_i+1 ||| f_del <<< M_i+1}           (12)
pair(a_i,x,b_j,y) = x ∪ y ∪ {A_i = f_ins <<< a ~~~ A_i ||| M_i}        (13)
         ∪ {M_i = f_PK <<< a ~~~ A_i+1 ~~~ b ~~~ M_j+1}                 (14)
         ∪ {M_i = f_Lr <<< a ~~~ A_i+1 ~~~ M_j+1}                       (15)
         ∪ {M_i = f_lR <<< M_i+1 ~~~ b ~~~ A_j+1}                       (16)
         ∪ {M_i = f_bg <<< M_i+1 ~~~ M_j+1}                             (17)
```

Compared to our use of the same signature with (the non-specialized) $G5$, all scoring functions take i as an implicit parameter, so calls to (say) f_{del} from different positions may be trained to assign different probabilities.

Non-ambiguity of generated models: We want to prove next that our model generator $G5M(genCM, C)$ generates, for every consensus structure C, a family model grammar which is unambiguous with respect to the trace semantics. The proof consists of two theorems:

Theorem 2 *Grammar $G5M$ is unambiguous with respect to the trace semantics.*

We might strive for an inductive proof of this theorem, but since we already have all the necessary machinery in place, we use an automated proof technique.

From $G5M$ we construct $\widehat{G5M}$ as explained in Section II-C. We have already observed that its derived alignments comply with the insert-before-delete-convention. Therefore, the generated alignments in fact denote traces. Remember that $G5M$ generates the same model-query alignment several times if and only if $\widehat{G5M}$ is syntactically ambiguous. We replace the fancy, two-character columns by single character encodings according to the following table:

$\overset{*}{\bullet}$	$\overset{-}{\bullet}$	$\overset{*}{\_}$	$\overset{<}{(}$	$\overset{>}{)}$	$\overset{<}{\bullet}$	$\overset{>}{\_}$	$\overset{<}{\_}$	$\overset{>}{\bullet}$	$\overset{<}{\_}$	$\overset{>}{\_}$
M	I	D	P	K	L	r	l	R	b	g

This turns $\widehat{G5M}$ into the grammar

$$
\begin{aligned}
A &\rightarrow \text{"}I\text{"}\ A \mid M && (18)\\
M &\rightarrow \varepsilon && (19)\\
M &\rightarrow \text{"}M\text{"}\ A \mid \text{"}D\text{"}\ M && (20)\\
M &\rightarrow \text{"}P\text{"}\ A\ \text{"}K\text{"}\ A && (21)\\
M &\rightarrow \text{"}L\text{"}\ A\ \text{"}r\text{"}\ M && (22)\\
M &\rightarrow \text{"}l\text{"}\ M\ \text{"}R\text{"}\ A && (23)\\
M &\rightarrow \text{"}b\text{"}\ M\ \text{"}g\text{"}\ M && (24)
\end{aligned}
$$

which is proved unambiguous by the *acla* ambiguity checker [3].
Q.E.D.

We can now show that by the generation algorithm, semantic non-ambiguity is inherited from $G5$ to the family model grammars.

Theorem 3 *Covariance models generated from a consensus structure C by $G5(genCM, C)$ are semantically non-ambiguous under the trace semantics.*

We note the following facts:

1) $\widehat{G5M}$ is syntactically non-ambiguous (Theorem 2).
2) Each derivation in $G5M$ describes an alignment of a query against *some* model.
3) By construction, all these alignments observe the insert-before-delete convention.
4) Any derivation in a generated model grammar $G5M_C$ can be mapped to a derivation in $G5M$. This is achieved by applying, for each production from $G5M_C$, the corresponding production without the subscripts form $G5M$. This means that all derivations $G5M_C$ also observe the insert-before-delete convention.
5) This mapping is injective. This holds because we can uniquely reconstruct the positional indices to turn a $\widehat{G5}$ derivation back into a $\widehat{G5_C}$ derivation, by keeping track of the number of symbols from $\{M, D, P, K, L, l, R, r, b, g\}$ generated so far (but not counting I).
6) Hence, if $G5M_C$ was ambiguous, $G5M$ would also be ambiguous, in contradiction to point (1).

Altogether, if there was a trace that had two different derivations in $G5M_C$, it would also have two different derivations in $G5M$. This is impossible according to point (1). Hence, a model grammar $G5M_C$ generated by $genCM$ is always non-ambiguous with respect to the trace semantics.
Q.E.D.

The correctness proof for the model generator here crucially depends on the non-ambiguity of the prototype grammar. When a prototype grammar G is ambiguous, a sophisticated generator can still avoid ambiguity in the generated models! However, in this case a proof might be difficult to achieve. If it fails, we can still convert each generated model G_C into the corresponding $\widehat{G_C}$, which can be submitted to ambiguity checking. This is the situation we will encounter when turning towards the "real-world" models in Rfam. There, we have an ambiguous prototype grammar and a sophisticated generation process, which makes it hard to prove properties about. Therefore, we next equip ourselves with an automated pipeline for ambiguity checking of Rfam models.

V. The ambiguity checking pipeline

Our ambiguity checking pipeline consists of three successive stages, named *cm2adp*, *adp2cfg*, and *acla*.

cm2adp: Upward compilation of Infernal generated covariance models: The upward compiler *cm2adp* accepts as input the table encoding a covariance model generated by *Infernal*. It translates it into the constituents of a mathematically equivalent ADP algorithm – a tree grammar, a signature, and an implementation of the stochastic scoring algebra using the parameters generated by *Infernal*. Once available in this form, additional evaluation algebras can be used in place of or jointly in products with the stochastic scoring algebra. Such semantic enrichment was the main purpose of developing *cm2adp*, and its scope

will be described in a forthcoming paper. One of these applications is the evaluation of the search space under a canonical mapping algebra, as we do here.

adp2cfg: Partial evaluation of grammar and canonical mapping algebra: The *adp2cfg* program is a simple utility implemented by Peter Steffen subsequent to [16]. It accepts a tree grammar G and a canonical mapping algebra A, such that a call to $G(A, x)$ for some query x, would enumerate all the members of the search space (i.e. all parses) under the canonical string mapping. Provided that the algebra A is very simple and uses only string constants and concatenation, *adp2cfg* succeeds with partial evaluation to produce the context free (string) grammar $\widehat{G}$ suitable for ambiguity checking according to Theorem 1.

acla: Ambiguity checking by language approximations: The *acla* phase simply calls the ACLA ambiguity checker for context free grammars, which is based on the recent idea of ambiguity checking via language approximations [3]. It has been used before, for example, on the grammar designed by Voss for probabilistic shape analysis of RNA [21]. Accumulating probabilities from the Boltzmann distribution of structures depends, just like stochastic scoring, critically on semantic non-ambiguity.

Due to the undecidability of the ambiguity problem, there is no guarantee that the *acla* phase will always return a definite answer. It may be unable to decide ambiguity for some covariance models. However, since the covariance models are larger, but less sophisticated than the grammar by Voss, we are confident that the formal undecidability of ambiguity will not be a practical obstacle in our context.

The overall pipeline: As all family model grammars derived from the same prototype grammar use the same signature, the evaluation algebra implementing the canonical mappings for the structural and the alignment semantics, CAN_{struct} and CAN_{align}, is the same for all, as described above. Let M denote a covariance model generated by *Infernal* from consensus structure C, given in *Infernal*'s tabular output format.

Let $(G_C, PROB) = \textit{cm2adp}(M)$ be the ADP program equivalent to M, generated by upward compilation.

Let $\widehat{G_{C,S}} = \textit{adp2cfg}(G_C, CAN_S)$ be the context free grammar generated by partial evaluation, where CAN_S is either CAN_{struct} or CAN_{align}.

Then, $\textit{acla}(\widehat{G_{C,S}}) \in \{YES, NO, MAYBE\}$ demonstrates semantic ambiguity or non-ambiguity of M with respect to the semantics S.

The trace semantics cannot be handled by *adp2cfg* because the recursive auxiliary function $\triangleright$ in CAN_{trace} can only be eliminated with an inductive argument. To demonstrate (non-)ambiguity with respect to the trace semantics, one shows (non-)ambiguity with respect to the alignment semantics plus (non-)observance of a uniqueness constraint such as the insert-before-delete convention. We now proceed to apply this pipeline.

VI. Semantics of Rfam family models

A. Model construction with Infernal

In this section, we look at covariance models as generated by *Infernal*. The difficulty here is that the prototype grammar is ambiguous and we do not have a fully formal specification of the generation algorithm. In order to create some suspense, we start with two quotations. The original publication [8] of 1994 states:

> "...we make the Viterbi assumption that the probability of the model emitting the sequence is approximately equal to the probability of the single best alignment of model to sequence, rather than the sum of all probabilities of all possible alignments. The Viterbi assumption conveniently produces a single optimal solution rather than a probability distribution over all possible alignments."

This points at an alignment or a trace semantics. In a more recent update, the *Infernal* Manual [14] touches on the issue of semantic ambiguity in the description of the model generation process, stating:

> "This arrangement of transitions guarantees that (given the guide tree) there is unambiguously one and only one parse tree for any given individual structure. This is important. The algorithm will find a maximum likelihood

> parse tree for a given sequence, and we wish to interpret this result as a maximum likelihood structure, so there must be a one-to-one relationship between parse trees and structures."

This seems to aim at a structure semantics, but since the same structure can always be aligned to the consensus (alias the "guide tree") in many ways, there *must* always be several parses for it, the scores of which should accumulate to obtain the likelihood of the structure.

Infernal starts from an initial multiple sequence alignment and generates models in an iteration of consensus estimation, model generation, and parameter training. Here we are concerned with the middle step, model generation from a given (current) consensus. The family consensus structure C is determined with an ambiguous grammar, parsing the multiple alignment and maximizing a mutual information score, and then one optimal parse (out of many) is fixed as the "guide tree". (In our construction, when C is given, this is simply the unique parse of C with tree grammar $G5$.) This guide tree is then used to generate productions by specializing the following prototype grammar:

Grammar $G_{infernal}$ taken from the *Infernal* manual [14]:

State type	Description	Production	Emission	Transition
P	(pair emitting)	$P \to aYb$	$e_v(a,b)$	$t_v(Y)$
L	(left emitting)	$L \to aY$	$e_v(a)$	$t_v(Y)$
R	(right emitting)	$R \to Ya$	$e_v(a)$	$t_v(Y)$
B	(bifurcation)	$B \to SS$	1	1
D	(delete)	$D \to Y$	1	$t_v(Y)$
S	(start)	$S \to Y$	1	$t_v(Y)$
E	(end)	$E \to e$	1	1

Here, Y is any state[8] chosen from the nonterminal symbols (state types) in the leftmost column. One recognizes the rules of the ambiguous $G1$ in the guise of $\{P \to aYb, L \to aY, R \to Ya, B \to SS, E \to \varepsilon\}$. The ambiguity inherent in a rule like $S \to SS$, parsing SSS both as $(SS)S$ and $S(SS)$ is *not* a problem in model generation, because the specialized rules $S_i \to S_j S_k$ are always unambiguous. However, insertions can be generated both from L and R, possibly competing for the generation of the same unpaired residues in the query.

$G_{infernal}$ is not really the complete prototype grammar in our sense, as rules for partial matches of base pairs in the consensus need to be added in the generation process. Overall, the generation method appears too complicated to strive for a formal proof of non-ambiguity of the generated models.

B. Checking Rfam models

We have checked 30 models from Rfam, the 15 smallest models with and without a bifurcation in their consensus structure, respectively. Model names and their consensus structures are listed in the appendix. Here, we give a resume of our findings:

Theorem 4 *In general, Rfam models are ambiguous with respect to the structure semantics. They do not assign a most likely structure to the query.*

This can be seen from testing with our pipeline, but is also easily seen by inspecting the generated models. Actually, alignments (1) and (2) in Section III-C are already an example of ambiguity with respect to the structure semantics, though only in principle, as they are not Rfam models. The explanation is that although *Infernal* takes care that the structural ambiguity of the prototype grammar does not enter the model grammar, it does not compensate for the fact that the same structure (assigned to the query) is aligned to the model in many ways. Hence, the score accounts for the structure associated with the optimal alignment, which need not be the highest scoring structure.
Q.E.D.

[8]The description in [14] uses a mixture of SCFG and HMM terminology.

Theorem 5 All tested Rfam models are non-ambiguous with respect to the alignment semantics. There is no evidence that this result should not carry over to Rfam models in general.

This observation was proved for some of the smallest models via producing their grammar $\widehat{G}$ and submitting it to the ambiguity checker. For larger models, the $ACLA$ checker ran out of resources. We applied some surgery by reducing successive stretches of either unpaired or paired residues (in the model) to stretches of at most three such residues. This is correct as it has already been tested that the rules within such stretches do not lead to ambiguity. After such surgery, the $ACLA$ checker succeeded for all models except Rf00161 and Rf00384.

For these two models, we resorted to a checking technique (rather than a proof) by use of a non-ambiguous reference grammar, as suggested in [16]: if we have a reference grammar R which generates non-ambiguously the alignments of a query to the given model, then we can compare the *number* of alignments produced by both grammars for a given input length[9]. The enormous size of the search space provides strong evidence that, if the number of alignments considered by either grammar coincides, the tested model grammar is also unambiguous. To apply this technique, we implemented a second $G5$-based model generator to generate family model grammars that are unambiguous for the alignment semantics. Let us call them G5.Rf00161 and G5.Rf00384. We then checked, using an evaluation algebra $COUNT$ which simply counts the number of solutions generated, for sequences x of various lengths that $Rf00161(COUNT, x) = G5.Rf00161(COUNT, x)$ and $Rf00384(COUNT, x) = G5.Rf00384(COUNT, x)$. For example, the value for $|x| = 10$ is 357,718,985,217,153 (Rf00161) and 261,351,290,279,573 (Rf00384). For $|x| = 20$, it is 774,380,024,914,343,603,750,401 (Rf00161) and 416,290,325,523,207,008,752,681 (Rf00384), computed independently by both models.
Q.E.D.[10]

The positive result that Rfam models correctly implement the alignment semantics is quite remarkable, given the notorious ambiguity introduced by the rules of $G1$, such as $S \rightarrow SS$ or $S \rightarrow aS|Sa$. This is achieved by details of the *Infernal* implementation. Applications of $S \rightarrow SS$ are made unambiguous by the use of the "guide tree", effectively choosing one of the many possible derivations of the consensus structure. Ambiguity effects of $S \rightarrow aS|Sa$ are avoided by disabling one of the alternatives in certain situations. Last not least, for searching with a model, *Infernal* has recently switched to using the Inside rather than the CYK algorithm [15], which changes the scoring but bypasses eventual ambiguity problems. However, for optimally aligning a sequence to the model, and hence also for model building, the CYK algorithm is still required. We will return to the use of the Inside algorithm in the conclusion.

Theorem 6 In general, Rfam models are ambiguous with respect to the trace semantics.

This is implied by our previous observations, as a trace corresponds to many alignments.
Q.E.D.

We also wondered whether the Rfam models could be tweaked to compute the trace semantics rather than the alignment semantics, simply by disabling some of the generated transitions (and re-training the parameters). Our upward compilation allows us to to eliminate certain transitions. We have been able to reduce the number of alignments considerably, but we have not found a way to reduce it to the number of traces.

C. A synopsis on RF00163 and RF01380

To give an impression of the degree of ambiguity observed with respect to structure and trace semantics, we compute some data for RF00163 and for RF01380, which are currently the smallest Rfam models

[9]Note that the number of alignments only depends on the length of model and query, but not on the concrete query sequence, and not on the grammar which implements the model.

[10]Strictly, this is not proved but only tested for Rf00161 and Rf00384 , but note that by throwing more computational ressources at the problem, we can prove the remaining candidates nonambiguous. For practical concerns, and with an eye on the other models not explicitly studied here, a quick check by the counting method is more appropriate.

with and without a bifurcation[11]. When counting structures for $n = 31$, computational resources were exhausted, as in contrast to traces and alignments, we have no polynomial time algorithm for counting structures – for the same reasons as discussed with ambiguity compensation.

Model	RF00163	RF01380
consensus	<<<<<<<*******<<<<<<*******>>>>>>>*******<<<<<>>>>>*>>>>>>	<<<<<<<<<*****>>>>>*>>>>
length (size)	45 (31)	19 (12)
$\|x\| = 12$		
structures	8,958	2,048
traces	35,330,137,025	141,120,525
alignments	715,114,268,248,121	35,330,137,025
$\|x\| = 31$		
structures	n.a.	n.a.
traces	1,916,865,700,889,408,588,480	30,405,943,383,200
alignments	1,862,835,338,733,448,037,194,650,687	208,217,738,981,165,823

The numbers of structures, traces and alignments are properties of the search space of a model/query alignment, independent of which algorithm traverses this search space. They have been computed twice, and they perfectly agree between the upward compiled *Infernal* models resulting from *cm2adp* and the models generated by our method from $G5M$. This can be taken as a strong indication that the two approaches are equivalent in terms of the search spaces they create. However, different grammars lead to different numbers of parameters and will, in general, not be equivalent as stochastic models.

D. Ambiguity compensation, revisited

Can we, given the trained and upward-compiled Rfam models, which compute the log-likelihood score according to the alignment semantics, use the same models with an accumulating scoring function to obtain the trace or the structure semantics? Mathematically, yes. What was explained in Section II-D for the simple stochastic context free grammars generally holds for all dynamic programming algorithms which can be expressed in the ADP framework, and hence also for our model grammars: Given grammar G, a scoring algebra S and an algebra CAN (i.e. CAN_{struct} or CAN_{trace}), just replace the maximizing objective function of S by an accumulating function, yielding scoring algebra S_{acc}. Then call $G(CAN * S_{acc}, x)$. The product algebra $CAN * S_{acc}$ maps all candidates to their equivalence classes under the canonical mapping, and performs accumulating scoring per class. Given the ADP machinery, which provides a generic implementation of the algebra product, ambiguity compensation comes without reprogramming efforts – but only in principle.

There are two obstacles to this elegant solution:

- In case of stochastic modeling, rather than multiplying probabilities which tend towards zero as the structure gets larger, one prefers to add their logarithms. Due to the monotonicity of the log function, the maximal log score still indicates the answer with maximum probability. However, substituting maximization by accumulation, one needs to compute and add the probabilities, potentially creating numerical problems.
- Efficiency of computing with $G(CAN * S_{acc}, x)$ depends on the number of canonical objects constructed, and as this number (in the case of covariance models) is exponential in the length of the query, this is practical only for very short sequences and small models. The implementation by a product algebra will have efficiency of $O(\alpha^n n^4)$, where α may be close to 1, but probably not close enough. Our counting results on the small models RF00163 and RF01380 indicate this.

[11]The number of structures without a bifurcation for $|x| = n$ is bounded from above by 2^{n-1}. The bound is sharp when $n < 2(p + 1)$, where p is the number of base pairs in the model. For larger n, the restriction that the query cannot be assigned more base pairs than the consensus becomes effective. This is why we see a 2^{11} for $n = 12$ and still cannot compute the exact number for $n = 31$.

Semantics	Direct computation in $\mathcal{O}(n^4)$ with	Computation by ambiguity compensation in $\mathcal{O}(\alpha^n \cdot n^4)$ with
alignment	$G_{\text{ali}}(\text{PROB}, q)$	—
trace	$G_{\text{trace}}(\text{PROB}, q)$	$G_{\text{ali}}(\text{CAN}_{\text{trace}} * \text{PROB}_{\text{acc}}, q)$
structure	—	$G_{\text{ali}}(\text{CAN}_{\text{struct}} * \text{PROB}_{\text{acc}}, q)$ $G_{\text{trace}}(\text{CAN}_{\text{struct}} * \text{PROB}_{\text{acc}}, q)$
sequence	$G_{\text{ali}}(\text{PROB}_{\text{acc}}, q)$ $G_{\text{trace}}(\text{PROB}_{\text{acc}}, q)$	$G_{\text{ali}}(\text{CAN}_{\text{seq}} * \text{PROB}_{\text{acc}}, q)$ $G_{\text{trace}}(\text{CAN}_{\text{seq}} * \text{PROB}_{\text{acc}}, q)$

TABLE I

THE SEMANTIC HIERARCHY. WE INDICATE GRAMMARS AND EVALUATION ALGEBRAS USED FOR EACH TASK. n IS THE LENGTH OF THE QUERY q. α DENOTES THE BASE OF THE EXPONENTIAL FACTOR, WHICH IS INCURRED WITH AMBIGUITY COMPENSATION. α DECREASES FROM TOP TO BOTTOM; FOR THE SEQUENCE SEMANTICS, $\alpha = 1$, AND BOTH COLUMNS DESCRIBE THE COMPUTATION VIA THE INSIDE ALGORITHM, USED WITH EITHER GRAMMAR.

VII. CONCLUSION

A. Summary of results

We have studied the problem of generating non-ambiguous family models from consensus structures. We clarified the notion of a semantics for family model grammars, and found that there are three well motivated, alternative definitions: the structure, the trace and the alignment semantics.

We developed the generation algorithm for the trace semantics, which, to our knowledge, has not been studied before. Along the way, we found a nice explanation of the prototype grammar as a grammar that allows for an infinite set of derivations, describing the alignment of the query to *all* possible models. The generation process can then be described lucidly by an evaluation algebra ($genCM$), which allows, for example, for a proof of non-ambiguity of the generated models.

For a summary of the semantic hierarchy, let us introduce yet another semantics. The *sequence semantics* assigns to each model/query alignment the same object of interest – the aligned query sequence itself. The canonical mapping algebra CAN_{seq} is trivial and left to the reader. Ambiguity compensation with respect to this mapping means summing up probabilities of *all* model/query alignments – this is commonly known as the Inside algorithm! According to this view, our intermediate semantic levels of trace and structure semantics can, alternatively, be viewed as as intermediates between CYK and Inside scoring, governed the equivalence relation induced by the canonical mapping. This view is summarized in Table 1. Note the lack of a grammar G_{struct}, which would allow for the polynomial-time computation of the structure semantics.

On the practical side, we have implemented the upward compilation of *Infernal* generated models to ADP. Here this compilation was used for connecting the Rfam models to our ambiguity checking pipeline. The upward compiled models, however, have other applications of interest, which will be described in a forthcoming study. But still, upward compilation from automatically generated tables is an ad-hoc measure, and in the long run, one might consider producing ADP code for the models directly when generated.

Also on the practical side, we have observed that the models generated from $G5M$ are relatively small. To extend the comparison, we have also implemented a $G5$-based generator for (provably) unambiguous family model grammars and the alignment semantics. Applying both our generators to Rf00163 and Rf01380, we can give concrete examples of the blow-up factor k (cf. Section III-A). We evaluate the size of the generated grammars.

Model	Model length/size	Rfam rules/nonterminals	$G5$ (alignment) rules/nonterminals	$G5$ (trace) rules/nonterminals
Rf00163	45 / 31	617 / 139	151 / 46	182 / 77
Rf01380	19 / 12	282 / 59	66 / 20	78 / 32

The factor (number of rules/model length) affects the runtime as a constant factor. It is about 14 for the Rfam models, 3.4 for the models derived from $G5$ with alignment semantics, and 4.1 for $G5M$-derived

models with the trace semantics. These factors cannot be directly compared, as *Infernal* implements affine gap scoring and some other features not included in the $G5$-based models. The factor (number of nonterminals/model size) measures the space requirements, as each nonterminal leads to a dynamic programming table. Here, the respective factors are 4.6, 1.5, and 2.5, approximately.

B. Directions of future research

We shortly sketch some research questions which are raised by our findings.

Investigation of the trace semantics: The trace semantics is new; it can be efficiently computed, and possibly, the performance of covariance models can be improved. Such an improvement is likely especially with respect to remote family members. This is because, when model and query have about the same length, one is likely to find a high-scoring alignment without adjacent deletions and insertions, whose score is not falsely reduced by ambiguity. Remote family members may require more insertions and deletions, some of them adjacent, and ambiguity strikes on a scale which is exponential in the number of such situations. With an eye on the use of the alignment semantics with Rfam, this implies that good scores can be taken as a strong indication of family membership, while low scores must be interpreted with care, especially when model and query significantly differ in length.

Investigation of the structure semantics: The structure semantics has been used so far with simple SCFGs, but not with family model grammars. The structure semantics seems appropriate when the goal is to use the information in the family model to assign a concrete, most likely structure to the query. This structure would have to be experimentally probed in order to verify that the query performs the same function as other family members.

However, in contrast to simple SCFGs, we do not know an efficient method to compute this semantics for family model grammars. Ambiguity compensation, as shown above, suffers from a runtime complexity dependent on the number of structures, which in turn grows exponentially with the sequence length. Efficient computation of the structure semantics is an interesting open challenge, where one must be aware that a polynomial time, exact algorithm may not exist. An ideal modeling tool would allow the user to specify the intended semantics, either at model generation time or when starting a search.

Smaller and faster models: The smaller size and better speed of models derived from a small grammar such as $G5$ deserves further study. Its use may have been discouraged by the diagnosis of the "abysmal" performance of $G5$ reported in [5]. Dowell and Eddy explain this performance by the overloading of rules:

> "The compact grammar $G5$, for instance, must invoke the same bifurcation rule $S \rightarrow aS\hat{a}S$ for every base pair and for every structural bifurcation, which are quite different structural features that occur with very different frequencies. The productions of $G5$ are thus "semantically overloaded": they collapse too many different types of information into the same parameters."

This explanation, appropriate as it is for simple SCFGs, also points to a remedy for the case of family model grammars. These grammars have position-specialized productions, and unless we tie parameters together irrespective of their structural position in the model, we can still train different and adequate parameters for different features. This requires careful engineering and empirical testing, but small grammars are still in the race. Note also that filtering techniques, which have been developed to speed up the present *Infernal*-generated models, can also be adapted to models generated from a different prototype grammar.

Comparing the performance of different prototype grammars: Dowell and Eddy diagnosed superior performance of another unambiguous SCFG ($G6$ which stems from Pfold [13]). However, this grammar was not tested as the prototype for model grammar generation. Given our compact algorithm of model generation – the generator from $G5$ is but 164 lines of ADP code – it maybe a justifiable effort to extend the Dowell and Eddy study to different model generators, training family models rather than simple SCFGs. We conjecture that our proof of a correct implementation of the trace (or the alignment) semantics could be adapted for a new family model generator, as long as an unambiguous prototype grammar is used. If not, there is still our ambiguity checking pipeline, which can be used to show correctness of the individual models after their generation.

ACKNOWLEDGMENTS

We acknowledge inspring discussions we had with many participants of the 2009 Benasque Workshop on Computational Methods for RNA Analysis, organized by E. Rivas and E. Westhof, where these ideas were first presented in public. In particular, Sean Eddy and Eric Nawrocki provided insight into model construction in *Infernal* and the underlying rationale. The authors thank Peter Steffen for implementing *adp2cfg* and Stefan Janssen for a careful reading and valuable comments. We also thank Ivo Hofacker for comments, help and support. This work has been funded, in part, by the Austrian GEN-AU project bioinformatics integration network III.

APPENDIX

Rfam models tested for non-ambiguity

15 smallest models from Rfam without a bifurcation:

```
RF00032.cm   ******<<<<<<*****>>>>>>>>>>
RF00037.cm   <<<<<**********************>>>>>*>>>>>
RF00180.cm   <<*<<<<********>>>>>>>>>>>>>>>>>>>>>>
RF00390.cm   *********<<<<<<<<<*>>>>
RF00453.cm   <<<<<<<<<<<<<<<<<<<<<<>>>>>>>>>>
RF00502.cm   *****<<<<<<<<<<********>>*>>>>>>>>>>
RF01072.cm   <<<<**********>>>>>>>>>>>>>>
RF01080.cm   *********<<<<<<<<<<<<<<<<<<>>>>>>>
RF01081.cm   <<<<<*********>>>>>>>>>>
RF01082.cm   ******<<<<<<<<<<********>>>>>>>
RF01112.cm   ******<<<<<<*******>>>>>
RF01115.cm   <<<********>>>>>>>>>>
RF01380.cm   <<<<<<<<<<*****>>>*>>>>
RF01381.cm   ***<<<<<<<<<<*>>>>>>>>>>
RF01382.cm   ***<<<<<<<<<<*>>>>>>>>
```

15 smallest models from Rfam with a bifurcation:

```
RF00008.cm   *<<<<<<<<<<<<<***>>>>>>>>>>*<<<<<<<<<*>>>>>>*>>>>>>*
RF00057.cm   **********<<<<<<<<<<*********>>*>>>>>>>>>>*<<<<<<<<<<*>>>>>>>>>>>>>>>
RF00161.cm   <<<<**********<<<<<<*********>>>>>>>>>>*<<<<<<<<<<*>>>>>>*>>>>>>*>>>>
RF00163.cm   <<<<<<<<<<<<<<<<<<<<<*>>>>>>>>>>*<<<<>>>>*>>>>>>
RF00184.cm   **********<<<<<<<<<<*>>>>>>>>>*<<<<<<<<<<<<<<<<<<*>>>>
RF00192.cm   <<<<<<<<<<<<<*>>>>>>>>>>*<<<<<<<<<<<<<<<<<<<<<<*>>>>>>
RF00384.cm   *<<<<<<<<<<<<*>>>>>>>>>>*<<<<<<<<<<*<<<<<<<<<*>>>>>>>>>>*>>>>>>>>>>>>>
RF00454.cm   **********<<<<<<<<<<*>>>>>>>>>>*<<<<<<<<<<*<<<<<<<<<<*>>>>>>>>>>*>>>>>>>>>>>
RF00517.cm   **********<<<<<<<<<<*>>>>>>>>>>*<<<<<<<<<<*<<<<<<<<<<*>>>>>>>>>>>
RF00630.cm   **********<<<<<<<<<<*>>>>>>>>>>*<<<<<<<<<<*<<<<<<<<<<*>>>>>>>>>>>>>>>>>>>>>*
RF00681.cm   <<<<<<<<<<<<<<<<<<<<<<*>>>>>>>>>>*<<<<<<<<<<*>>>>>>>>>>*<<<<<<<<<<*>>>>>>>>>>
RF01068.cm   <<<<<<<<<<*>>>>>>>>>>*<<<<<<<<<<*<<<<<<<<<<*>>>>>>>>>>
RF01116.cm   <<<<<<<<<<*>>>>>>>>>>*<<<<<<<<<<*<<<<<<<<<<*>>>>>>>>>>*>>>>>>>>>>
RF01388.cm   *<<<<<<<<<<<<<<<<<<<<*>>>>>>>>>>*<<<<<<<<<<*>>>>>>>>>>*>>>>>>>>>>
RF01403.cm   <<<**********<<<<<<<<<<*>>>>>>>>>>*<<<<<<<<<<*>>>>>>>>>>*<<<<<<<<<<*>>>>>>>>>>
```

REFERENCES

[1] P Baldi and S Brunak. *Bioinformatics – the machine learning approach*. MIT Press, 1998.

[2] A Bateman, E Birney, L Cerruti, R Durbin, L Etwiller, S R Eddy, S Griffiths-Jones, K L Howe, M Marshall, and E L Sonnhammer. The Pfam protein families database. *Nucleic Acids Res*, 30(1):276–280, Jan 2002.

[3] Claus Brabrand, Robert Giegerich, and Anders Møller. Analyzing Ambiguity of Context-Free Grammars. In *Proc. 12th International Conference on Implementation and Application of Automata, CIAA '07*, volume 4783 of *LNCS*. Springer-Verlag, July 2007. Extended version submitted for journal publication.

[4] Broňa Brejová, Daniel G. Brown, and Tomáš Vinař. The most probable annotation problem in HMMs and its application to bioinformatics. *Journal of Computer and System Sciences*, 73(7):1060–1077, March 2007.

[5] R D Dowell and S R Eddy. Evaluation of several lightweight stochastic context-free grammars for RNA secondary structure prediction. *BMC Bioinformatics*, 5:71–71, Jun 2004.
[6] R Durbin, S Eddy, A Krogh, and G Mitchison. *Biological Sequence Analysis*. Cambridge University Press, 2006 edition, 1998.
[7] S R Eddy. Profile hidden markov models. *Bioinformatics*, 14(9):755–763, 1998.
[8] Sean R. Eddy and Richard Durbin. RNA sequence analysis using covariance models. *Nucleic Acids Res*, 22(11):2079–2088, June 1994.
[9] P P Gardner, J Daub, J G Tate, E P Nawrocki, D L Kolbe, S Lindgreen, A C Wilkinson, R D Finn, S Griffiths-Jones, S R Eddy, and A Bateman. Rfam: updates to the RNA families database. *Nucleic Acids Res*, 37(Database issue):136–140, Jan 2009.
[10] R. Giegerich. Explaining and Controlling Ambiguity in Dynamic Programming. In *Proc. Combinatorial Pattern Matching*, volume 1848 of *Springer Lecture Notes in Computer Science*, pages 46–59. Springer, 2000.
[11] R. Giegerich, C. Meyer, and P. Steffen. A Discipline of Dynamic Programming over Sequence Data. *Science of Computer Programming*, 51(3):215–263, 2004.
[12] J E Hopcroft and J D Ullman. *Formal languages and their relation to automata*. Addison-Wesely, 1969.
[13] B Knudsen and J Hein. Pfold: RNA secondary structure prediction using stochastic context-free grammars. *Nucleic Acids Res*, 31(13):3423–3428, Jul 2003.
[14] Sean Eddy Lab. *INFERNAL User's Guide. Sequence analysis using profiles of RNA secondary structure*, version 1.0 edition, January 2009. http.//infernal.janelia.org.
[15] E P Nawrocki, D L Kolbe, and S R Eddy. Infernal 1.0: inference of RNA alignments. *Bioinformatics*, 25(10):1335–1337, Mar 2009.
[16] Janina Reeder, Peter Steffen, and Robert Giegerich. Effective ambiguity checking in biosequence analysis. *BMC Bioinformatics*, 6:153, 2005.
[17] Eddy S. Hmmer user's guide. Technical report, Howard Hughes Medical Institute, 2003.
[18] Y Sakakibara, M Brown, R Hughey, I S Mian, K Sjölander, R C Underwood, and D Haussler. Stochastic context-free grammars for tRNA modeling. *Nucleic Acids Res*, 22(23):5112–5120, Nov 1994.
[19] D Sankoff and Kruskal J. *Time warps, string edits, and macromolecules*. Addison-Wesley, 1983.
[20] Peter Steffen and Robert Giegerich. Versatile and declarative dynamic programming using pair algebras. *BMC Bioinformatics*, 6(1):224, September 2005.
[21] B Voss, R Giegerich, and M Rehmsmeier. Complete probabilistic analysis of RNA shapes. *BMC Biol*, 4:5–5, 2006.
[22] M Waterman. *Introduction to computational biology*. Chapman & Hall, 1994.