An Earley-style Parser for Solving the RNA-RNA Interaction Problem

Bachelor Thesis

Supervisor: **Prof. Dr. Markus E. Nebel,**

head of "AG Algorithmen und Komplexität"

Technische Universität Kaiserslautern

Sebastian Wild

(Bachelorstudiengang Informatik)

February 4, 2011

Abstract: It has been observed that for understanding the biological function of certain RNA molecules, one has to study *joint* secondary structures of *interacting* pairs of RNA.

In this thesis, a new approach for *predicting* the joint structure is proposed and implemented. For this, we introduce the class of m-dimensional context-free grammars — an extension of stochastic context-free grammars to multiple dimensions — and present an Earley-style semiring parser for this class. Additionally, we develop and thoroughly discuss an implementation variant of Earley parsers tailored to efficiently handle *dense* grammars, which embraces the grammars used for structure prediction.

A currently proposed partitioning scheme for joint secondary structures is transferred into a two-dimensional context-free grammar, which in turn is used as a stochastic model for RNA-RNA interaction. This model is trained on actual data and then used for predicting most likely joint structures for given RNA molecules. While this technique has been widely used for secondary structure prediction of single molecules, RNA-RNA interaction was hardly approached this way in the past.

Although our parser has $\mathcal{O}(n^3m^3)$ time complexity and $\mathcal{O}(n^2m^2)$ space complexity for two RNA molecules of sizes n and m, it remains practically applicable for typical sizes if enough memory is available. Experiments show that our parser is much more efficient for this application than classical Earley parsers. Moreover the predictions of joint structures are comparable in quality to current energy minimization approaches.

Zusammenfassung: Es ist bekannt, dass die biologische Funktion bestimmter RNA-Moleküle von der Struktur abhängt, die das Molekül *gemeinsam* mit einem anderen RNA-Molekül formt, der sogenannten *joint secondary structure*.

Wir stellen in dieser Arbeit einen neuen Ansatz zu deren Vorhersage vor. Dazu führen wir die Klasse der m-dimensionalen kontextfreien Grammatiken ein — einer Erweiterung stochastischer kontextfreier Grammatiken auf mehrere Dimensionen — und stellen einen Semiring-Earley-Parser für diese Klasse vor. Außerdem entwickeln wir eine Implementierungsvariante für Earley-Parser, die speziell *dichte* Grammatiken effizient verarbeitet. Dazu zählen insbesondere die Grammatiken, die zur Strukturvorhersage verwendet werden.

Wir übersetzen ein aktuelles Partitionierungsschema für *joint secondary structures* in eine zwei-dimensionale kontextfreie Grammatik, die ein stochastisches Modell für gemeinsame Sekundarstrukturen impliziert. Dieses Modell trainieren wir anhand bekannter RNA-RNA-Interaktionen und sagen dann wahrscheinlichste Strukturen vorher. Diese Idee wurde bereits erfolgreich auf einzelne RNA-Moleküle angewandt, kaum jedoch auf interagierende RNA-Paare.

Obwohl unser Parser Laufzeit in $\mathcal{O}(n^3m^3)$ und Speicherplatz in $\mathcal{O}(n^2m^2)$ für zwei RNAs der Länge n und m benötigt, sind typische Eingabegrößen mit vertretbarem Aufwand vorhersagbar, sofern genug Arbeitsspeicher zu Verfügung steht. Experimente zeigen, dass unsere Earley-Implementierung für diese Anwendung deutlich effizienter arbeitet als klassische Earley-Parser. Des Weiteren ist die Qualität der Vorhersagen vergleichbar mit der aktueller Energie-Minimierungsansätze.

Eidesstattliche Erklärung

Ich versichere hiermit, dass ich die vorliegende Bachelorarbeit mit dem Thema "An Earley-style Parser for Solving the RNA-RNA Interaction Problem" selbstständig verfasst habe und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen, die anderen Werken dem Wortlaut oder dem Sinne nach entnommen wurden, habe ich durch die Angabe der Quelle, auch der benutzten Sekundärliteratur, als Entlehnung kenntlich gemacht.

(Ort, Datum) (Unterschrift)

Contents

1. Introduction

Ribonucleic acid (RNA) is a class of biomolecules that play a central role in protein synthesis. Many functions fulfilled by RNA molecules depend on the three-dimensional shape of the molecule. Each RNA is a single strand of nucleotides and different RNA molecules only differ in the used bases in these nucleotides. The three-dimensional shape is caused by *attraction* between different nucleotides, forcing the linear backbone to twist and fold in space.

Biological studies have revealed that for the function of some types of RNA in living beings, the three-dimensional form of RNA is very important. However, it is rather expensive and time-consuming to determine three-dimensional foldings in laboratory, whereas *sequencing*, i. e. determining the sequence of bases forming the backbone of an RNA, has become cheap and fast due to automation.

Consequently, computational approaches have been developed for *predicting* the spacial structure of RNA from the sequence. Typically, the model of *secondary structures* is used to simplify prediction: Each base in the primary structure, i. e. the sequence, can either be *paired* with exactly one other base or remains *unpaired*. Paired bases will be located near each other in space. Secondary structure does not fully represent all details of three-dimensional folding, but is able to model most important aspects thereof. So, predicting correct secondary structures is considered a valuable goal — chapter 10 of [DEKM98] provides an overview of approaches to it.

It has been observed that for understanding the biological function of certain RNA molecules, it is not enough to look at a *single* RNA and its secondary structure in isolation. One important aspect overlooked that way is the *interaction* of *two* RNA molecules. During protein biosynthesis such interactions can hinder mRNA leaving the nucleus, thereby controlling gene expression. Since RNA-RNA interaction complexes are hard to analyse in laboratory, some research was done, trying to computationally *predict* the joint secondary structure of such an interacting RNA complex from the primary structures.

Most of the proposed approaches (e. g. [AKN+06], [AZC05], [SBS09]) are generalizations of Zuker's algorithm for computing the secondary structure with minimal free energy. One problem is that energy contributions for more complicated types of loops — as the new ones possible when considering *two* RNAs — are not known.

In the recent past, there has been an approach to RIP using stochastic multiple context-free grammars — a generalization of stochastic context-free grammars — in [KAS09]. However, the grammar used by Kato et al. is rather simple and does for instance not offer the possibility to assign different probabilities to different types of substructures.

Therefore, we propose a new approach in the following, using a grammar from a similar class which provides a much more detailed model. For efficiently working with this grammar, we develop a suitable parser for our class of grammars, as well.

The rest of this work is organized as follows: Chapter 2 reviews concepts and definitions needed for our approach. Chapter 3 describes our new contributions for RNA-RNA interaction prediction. It introduces the grammar class of m-dimensional context-free grammars and gives a high-level view of our parser for those. Our implementation of this parser needs a few restrictions and features some interesting optimizations — which is discussed in detail in chapter 4. In chapter 5 we present jackRIP, our C++ implementation according to chapter 4, and evaluate our approach in an experiment with actual data. Finally, we summarize our work in chapter 6 and propose two promising extensions based on recent work in the field.

2. Basics

For the presentation of our approach, we need some terms, facts and background knowledge to build on. As typical for computational biology — and structure prediction in particular — mainly two fields are combined: The theory of formal grammars on the one hand and knowledge from molecular biology on the other hand.

A quick review of the essential formal models is given here, in order to create a common basis of notation. For further information, [HU79] provides a comprehensive discussion of formal language theory. For the biological background needed here, any textbook on molecular biology will suffice.

2.1. Context-free grammars

Any textbook on theoretical computer science contains a definition similiar to the following. We adhere in the main to [HU79], but use different names for the components of a grammar:

Definition: A **context-free grammar** (CFG) is a tuple $G = (N, \Sigma, R, S)$. The elements are

- N, a finite set of **non-terminals**,
- Σ, a finite set of **terminals**, also called the **alphabet**,
- $R \subseteq N \times (N \cup \Sigma)^\star$, a finite set of **rules/productions** and
- the **start symbol** $S \in N$.

We require N and Σ to be disjoint. A rule $(A, \gamma) \in R$ is written as $A \to \gamma$. For convenience, we write $R_A := \{A \to \gamma \in R\}$ for a non-terminal $A \in N$, i. e. R_A is the set of rules whose left hand side is A.

A word over $N \cup \Sigma$ is called **sentential form**. We define the derivation relation $\underset{G}{\Rightarrow} \subseteq (N \cup \Sigma)^\star \times (N \cup \Sigma)^\star$ on sentential forms by

$$\alpha A \beta \underset{G}{\Rightarrow} \alpha\gamma\beta \qquad \text{iff} \qquad A \to \gamma \ \in R\,.$$

If the grammar is clear from the context, we will dispose of the G. The reflexive and transitive closure of $\Rightarrow$ is written $\Rightarrow^*$, the transitive — but not reflexive — closure is denoted by $\Rightarrow^+$. The **language** $\mathcal{L}(G)$ generated by G is the set of all terminal words derivable in G, i. e. $\mathcal{L}(G) := \{w \in \Sigma^\star : S \underset{G}{\Rightarrow^*} w\}$. A sequence of derivation steps $\alpha_1 \Rightarrow \alpha_2 \Rightarrow \cdots \Rightarrow \alpha_k$ is called a **derivation**. A derivation is a **leftmost** derivation if in every step the replaced non-terminal is the leftmost non-terminal present. In the following we will only consider leftmost derivations.

Derivations can also be represented as ordered trees, where each node represents a non-terminal, a terminal or the empty string ε. Inner nodes always correspond to non-terminals and the labels of the children of a node in left-to-right order have to form the right hand side of a rule for the parent. There exists a bijection between leftmost derivations and derivation trees.

We call a CFG G **loop-free** if there is no derivation of the form $A \overset{+}{\underset{G}{\Rightarrow}} A$ for any $A \in N$. If there does not even exist a non-terminal A such that $A \overset{+}{\underset{G}{\Rightarrow}} A\alpha$, we call G **free of left-recursion**.

The **decision problem** for context-free grammars '$w \overset{?}{\in} \mathcal{L}(G)$' is the task "Given a terminal string w, determine whether w can be generated in G or not". Several efficient algorithms are known solving $w \overset{?}{\in} \mathcal{L}(G)$ in time $\mathcal{O}(|w|^3)$, e. g. the Cocke-Younger-Kasami algorithm (CYK) or Earley's algorithm.

The **parsing problem** for context-free grammars '$S \overset{?}{\Rightarrow} w$' is the task "Given w, decide whether $w \in \mathcal{L}(G)$ and if so determine a derivation (tree) for w". Simple additions to the mentioned algorithms allow to solve the parsing problem with negligible additional cost.

Two CFGs G_1 and G_2 are called **equivalent**, iff $\mathcal{L}(G_1) = \mathcal{L}(G_2)$. If for every word $w \in \mathcal{L}(G)$ exactly *one* leftmost derivation exists, the CFG G is called **umambiguous**, otherwise **ambiguous**.

2.2. Earley-Parsing

In [Ear70], Earley presented a new algorithm for efficiently solving the decision problem for context-free-grammars G. Although a zoo of different parsing strategies has been developed — see e. g. [GJ08] for a quite exhaustive collection — Earley's algorithm has some remarkable qualities:

- It can handle *arbitrary* CFGs — no normal form is required.
- Worst case complexity is $\mathcal{O}(|w|^2)$ memory and $\mathcal{O}(|w|^3)$ runtime, however for unambiguous grammars runtime is $\mathcal{O}(|w|^2)$ and for some large subclasses even linear (see [GJ08, section 7.2.1.3]).
- Earley parsers efficiently handle left-recursion (see [GJ08, section 7.2.5]).

Especially the first property is valuable, even more for stochastic grammars, where transformations to normal forms need to retain the stochastic model implied by the grammar.

We will give a very high-level description of Earley's parser, viewing recognition of an input word w as a deductive process in a properly chosen calculus. This makes it rather easy to show correctness, but lacks details of efficient implementation. Since the grammars we will use are quite different from typically studied ones for Earley parsing, usual implementations are not suitable. We discuss our design in detail in chapter 4.

The calculus of an Earley parser consists of **items**, which are tuples of position indices and **dotted rules**. A dotted rule is a rule of the grammar with dot-symbol $\bullet$ inserted *somewhere* in the right hand side of this rule — dots at the very left and right end are explicitly allowed. The dotted rules for $A \to bA$ are $A \to \bullet bA$, $A \to b\bullet A$ and $A \to bA\bullet$.

Assume we are parsing the input word $w \in \Sigma^n$ in a CFG $G = (N, \Sigma, R, S')$, with exactly one rule for S', namely $S' \to S$, where S is an arbitrary non-terminal $S \in N$, $S' \neq S$.[1] We will write items in the form

$$(i \;\; j, A \to \alpha \bullet \beta) \qquad \text{for } A \to \alpha\beta \in R \wedge 1 \leqslant i \leqslant j \leqslant n+1\,.$$

Every derivation in the calculus begins with the start item $I_{\text{start}} := (1 \;\; 1, S' \to \bullet S)$ and may be continued using the following three inference rules:

- **Predictor** $\dfrac{(i \;\; j, C \to \eta \bullet A\mu) \quad \text{Rule}\,[A \to \beta]}{(j \;\; j, A \to \bullet \beta)}$
- **Scanner** $\dfrac{(i \;\; j-1, A \to \alpha \bullet w_{j-1}\beta)}{(i \;\; j, A \to \alpha w_{j-1} \bullet \beta)}$
- **Completer** $\dfrac{(i \;\; r, A \to \alpha \bullet B\beta) \quad (r \;\; j, B \to \eta \bullet)}{(i \;\; j, A \to \alpha B \bullet \beta)}$

Theorem 1 (Correctness of Earley's algorithm)**:**
For CFG G and terminal word $w \in \Sigma^n$ as above, an arbitrary grammar rule $A \to \alpha\beta$ and indices $1 \leqslant i \leqslant j \leqslant n+1$, the following invariant holds:

$$(i \;\; j, A \to \alpha \bullet \beta) \text{ is derived} \quad \text{iff} \quad \exists \gamma \in (\Sigma \cup N)^\star \quad \begin{array}{lllll} S' & \overset{*}{\underset{G}{\Rightarrow}} & w_{1,i-1} & A & \gamma \\ & \underset{G}{\Rightarrow} & w_{1,i-1} & \alpha & \beta\,\gamma \\ & \overset{*}{\underset{G}{\Rightarrow}} & w_{1,i-1}\, w_{i,j-1} & \beta & \gamma \end{array} \,. \tag{2.1}$$

For a word w, $w_{s,s-1}$ denotes the empty string ε. Before we prove the theorem, let us harvest: For I_{start}, the right side of (2.1) is trivially true as only $S' \Rightarrow S$ remains. So beginning with the start item is legitimate. If we now assume, that S' does not occur in any right hand side — easily achieved by replacing it by S — we find that

$$I_{\text{goal}} := (1 \;\; n+1, S' \to S \bullet)$$

is derived if and only if $w \in \mathcal{L}(G)$, as $w_{1,0} = \varepsilon$ and γ can only be ε, as well. So, if we seed the deduction with I_{start}, theorem 1 states that $w \in \mathcal{L}(G)$ iff I_{goal} is derivable, thereby transferring the decision problem $w \overset{?}{\in} \mathcal{L}(G)$ into our calculus and proving the correctness of Earley's algorithm.

We are discussing the proof of theorem 1 in much detail in the following, since then, the correctness proof for our parser in section 3.4 will only require slight extensions to this proof. A few terms are handy, as they give an intuition about how Earley-parsing works.

[1] Obviously all CFGs can easily be converted into this form by adding a new axiom and the given rule. Being picky, this is a kind of "Earley normal form" — but we chose Earley because of *no need* for normal forms! Actually Earley parsing works for arbitrary grammars; requiring trivial additional steps before and after deduction process. The restriction only makes our presentation more compact, so we will stick to it. Furthermore, adding a new axiom with a single rule never changes implied stochastic models of a SCFGs, which is typically not the case for classical grammar transformations.

Definition: For $T = (V, E)$ an arbitrary partial parse tree, we define the **outline of T** $O \in (V \times V)^\star$ as the *sequence of edges* traversed in a tree traversal of T:

We start at the root, then visit all its children in left-to-right order. To 'visit' a node here means to append the edge from the root to the child, then treat the child recursively as the root of a subtree — possibly adding more edges — and finally add the edge from the child back to its parent, the root. This way we walk along the 'rim' of the tree.

O is obviously a closed path, so every pair of nodes v and u, where u is (direct) child of v, shows up exactly twice in O, once for descent as (v, u), once for ascent as (u, v).

Now, let $\mathcal{E}$ be a *prefix* of O, stopping with an occurrence of edge e. We call $\mathcal{E}$ an **Earley-walk**, if all *leaves* visited along $\mathcal{E}$ — except for u if $e = (v, u)$ and u is a leaf — are labelled with a terminal or ε. The concatenation of all these terminals in the order they are visited is called the **word of $\mathcal{E}$**, $word(\mathcal{E}) \in \Sigma^\star$. This simply means that no non-terminal lying in $\mathcal{E}$ was left unexpanded.

For an Earley-walk $\mathcal{E}$, we call $\big(i\ \ j\,,\ A \to \alpha \bullet \beta\big)$ the **item of the Earley-walk**, denoted by $item(\mathcal{E})$, if all of the following holds:

- The last edge in $\mathcal{E}$ is e and $e \in \big\{(v_A, u), (u, v_A)\big\}$ where A is the label of v_A and w. l. o. g. is v_A (direct) parent of u.
- $word(\mathcal{E}) = w_{1,j-1}$.
- The number of terminals visited by $\mathcal{E}$, which are *not* located in the subtree rooted by v_A is $i - 1$.[2]
- If $e = (v_A, u)$, i. e. we are descending from v_A to u:
  α is the concatenation of labels of children of v_A left of u *exclusively*, and β holds the labels of the remaining children starting *with* u.
- If $e = (u, v_A)$, i. e. we are ascending from u to v_A:
  α is the concatenation of labels of children of v_A left of u *inclusively*, and β holds the labels of the remaining children starting *behind* u.

Lemma 2: A partial derivation tree for $S' \Rightarrow^* w_{1,i-1}A\gamma \Rightarrow w_{1,i-1}\alpha\beta\gamma \Rightarrow^* w_{1,j-1}\beta\gamma$ with an Earley-walk $\mathcal{E}$ fulfilling $item(\mathcal{E}) = \big(i\ \ j\,,\ A \to \alpha \bullet \beta\big)$ exists for some $\gamma \in (\Sigma \cup N)^\star$, if and only if $\big(i\ \ j\,,\ A \to \alpha \bullet \beta\big)$ is derivable in the above calculus.

Proof: We show the implication '$\Rightarrow$' by induction on the length of Earley-walks: Let T be a (partial) derivation tree for $S' \Rightarrow^* w_{1,i-1}A\gamma \Rightarrow w_{1,i-1}\alpha\beta\gamma \Rightarrow^* w_{1,j-1}\beta\gamma$ for some $1 \leqslant i \leqslant j \leqslant n+1$ and $\gamma \in (\Sigma \cup N)^\star$ with Earley-walk $\mathcal{E}$ of length $|\mathcal{E}| = l$ such that $item(\mathcal{E}) = \big(i\ \ j\,,\ A \to \alpha \bullet \beta\big) =: I$.

For $l = 1$, the derivation can only be $S' \Rightarrow S$, so $I = I_{start}$, which is trivially derivable. So, the basis of induction is established.

For $l > 1$, we call $e := \mathcal{E}_{|\mathcal{E}|} \in \big\{(v_A, u), (u, v_A)\big\}$ the last edge in $\mathcal{E}$ and assume v_A is parent of u. Further be $\mathcal{E}' := \mathcal{E}_{1,|\mathcal{E}|-1}$ the Earley-walk resulting from $\mathcal{E}$ by deleting the last edge e and call e' the last edge in $\mathcal{E}'$. Now we distinguish cases:

[2] This leaves $j - i$ terminals for the part of the subtree rooted by v_A that is visited by $\mathcal{E}$; excluding u if $e = (v_A, u)$.

- $e = (v_A, u)$ and u is *not the leftmost* child of v_A:
  Call u' the left sibling of u. By definition of outline, the last edge in $\mathcal{E}'$ must be $e' = (u', v_A)$. This in turn implies $\mathit{item}(\mathcal{E}') = \mathit{item}(\mathcal{E})$ by definition of *item* because ascending edges *in*clude the child, whereas descending edges *ex*clude it. As $|\mathcal{E}'| < l$, applying induction hypothesis yields that $\mathit{item}(\mathcal{E}') = I$ is derivable.

- $e = (v_A, u)$ and u is the *leftmost* child of v_A:
  As no leaves in subtree v_A are visited by $\mathcal{E}$ — and u is explicitly excluded — we get $i = j$ and $\alpha = \varepsilon$, so $I = \big(j \;\; j\,,\; A \to \bullet\,\beta\big)$. The last edge in $\mathcal{E}'$ is $e' = (v_C, v_A)$ for v_C the parent of v_A. Let $C \in N$ be the label of v_C. As T is a derivation tree, the labels of all children of v_C form the right hand side of a rule $C \to \eta A \mu \in R$.

  Now, we remove in T all children of v_A and call the result T'. Obviously, T' is a partial derivation tree for $S' \Rightarrow^* w_{1,r-1} C \delta \Rightarrow w_{1,r-1}\eta A \mu \delta \Rightarrow^* w_{1,j-1} A \mu \delta = w_{1,j-1} A \gamma$ for some $1 \leqslant r \leqslant j$. The last edge in $\mathcal{E}$, e, was the first edge to enter subtree v_A, so up to e, all edges remain valid in T', as well. Consequently, $\mathcal{E}'$ is an Earley-walk in T', so $I' = \mathit{item}(\mathcal{E}') = \big(r \;\; j\,,\; C \to \eta \bullet A\mu\big)$ is derivable by induction hypothesis. Finally, $\frac{I' \;\; \text{Rule}[C \to \eta A \mu]}{I}$ is an instance of prediction rule, so I is derivable, as well.

- $e = (u, v_A)$ and u is a leaf:
  By definition, the label of u is not a non-terminal. If it is ε, we have $\mathit{item}(\mathcal{E}') = \mathit{item}(\mathcal{E})$, so $I = \mathit{item}(\mathcal{E})$ is derivable by induction hypothesis. If the label is a terminal, it must be $w_{j-1} = \alpha_{|\alpha|}$ by definition of T. But then, T is a partial derivation tree for $S' \Rightarrow^* w_{1,i-1} A \gamma \Rightarrow w_{1,i-1}\alpha_{1,|\alpha|-1}\alpha_{|\alpha|}\beta\gamma \Rightarrow^* w_{1,j-2}\;\alpha_{|\alpha|}\beta\gamma$, as well. As $e' = (v_A, u)$, we have $I' := \mathit{item}(\mathcal{E}') = \big(i \;\; j-1\,,\; A \to \alpha_{1,|\alpha|-1} \bullet w_{j-1}\beta\big)$, which is derivable by induction hypothesis. Finally, $\frac{I'}{I}$ is an instance of scanning, deriving I.

- $e = (u, v_A)$ and u is an inner node:
  The label of u must be a non-terminal, call it B. As T is a derivation tree, the children of u form a rule, say $B \to \eta$. $e' = (u', u)$, for u' the rightmost child of u. T may be seen as derivation tree for $S' \Rightarrow^* w_{1,i-1} A\gamma \Rightarrow w_{1,i-1}\alpha_{1,|\alpha|-1} B\beta\gamma \Rightarrow^* w_{1,r-1} B\beta\gamma \Rightarrow^* w_{1,j-1}\beta\gamma$ for some $i \leqslant r \leqslant j$. Put together, this means that $I' := \mathit{item}(\mathcal{E}') = \big(r \;\; j\,,\; B \to \eta \bullet\big)$ and I' is derivable by induction hypothesis.

  Now be T' the tree resulting from removal of all children of u in T . T' obviously is a derivation tree for $S' \Rightarrow^* w_{1,i-1} A\gamma \Rightarrow w_{1,i-1}\alpha_{1,|\alpha|-1} B\beta\gamma \Rightarrow^* w_{1,r-1}\; B\beta\gamma$. Let $\mathcal{E}''$ be the prefix of $\mathcal{E}$ up to the first occurrence of u in an edge, i. e. the last edge in $\mathcal{E}''$ is (v_A, u). Again, $\mathcal{E}''$ is an Earley-walk in T' as no edges in the subtree u show up in $\mathcal{E}''$. By induction hypothesis we get that $I'' := \mathit{item}(\mathcal{E}'') = \big(i \;\; r\,,\; A \to \alpha_{1,|\alpha|-1} \bullet B\beta\big)$ is derivable.

  Last, but not least, $\frac{I'' \;\; I'}{I}$ is an instance of the completion rule, completing the derivation of I.

For the other direction, we assume item $I = \big(i \;\; j\,,\; A \to \alpha \bullet \beta\big)$ is derivable in the calculus by at most l inference rule applications and show the claim by induction on l.

$l = 0$ implies $I = I_{\text{start}}$, and the derivation $S' \Rightarrow S$ together with the one-edge Earley-walk suffices, acting as basis of induction. For $l > 0$, there are three mutually exclusive cases, how item I was derived:

- $\alpha = \varepsilon$:
  This implies $i = j$, i.e. $I = \big(j \;\; j, A \to \bullet\beta\big)$. So, I must have been derived from some item $I_1 = \big(r \;\; j, C \to \eta \bullet A\mu\big)$ by prediction. Applying induction hypothesis to I_1, a derivation tree T' for $S' \Rightarrow^* w_{1,r-1}C\delta \Rightarrow w_{1,r-1}\eta A\mu\delta \Rightarrow^* w_{1,j-1}A\mu\delta = w_{1,j-1}A\gamma$ with Earley-walk $\mathcal{E}'$ exists such that $\mathit{item}(\mathcal{E}') = I_1$. By definition, $\mathcal{E}'$ ends with (v_C, v_A), where C is the label of v_C and v_A is child of v_C.[3] Appending child nodes for rule $A \to \beta$ at v_A in T' gives T, a derivation tree for $S' \Rightarrow^* w_{1,r-1}C\delta \Rightarrow w_{1,r-1}\eta A\mu\delta \Rightarrow^* w_{1,j-1}A\mu\delta \Rightarrow w_{1,j-1}\beta\mu\delta = w_{1,j-1}\beta\gamma$. Now, we can give an Earley-walk $\mathcal{E} := \mathcal{E}' \cdot (v_A, u)$ in T for u the leftmost new child satisfying $\mathit{item}(\mathcal{E}) = I$.

- $\alpha \neq \varepsilon \;\wedge\; \alpha_{|\alpha|} = w_{j-1} \in \Sigma$:
  This implies that I was derived from $I_1 = \big(i \;\; j-1, A \to \alpha_{1,|\alpha|-1} \bullet w_{j-1}\beta\big)$ by scanning. By induction hypothesis, T for $S' \Rightarrow^* w_{1,i-1}A\gamma \Rightarrow w_{1,i-1}\alpha_{1,|\alpha|-1}w_{j-1}\beta\gamma \Rightarrow^* w_{1,j-2}w_{j-1}\beta\gamma = w_{1,j-1}\beta\gamma$ exists with Earley-walk $\mathcal{E}'$ fulfilling $\mathit{item}(\mathcal{E}') = I_1$. (v_A, u) is the last edge in $\mathcal{E}'$ for u the $|\alpha|$th child of v_A.[4] Thus, $\mathcal{E} := \mathcal{E}' \cdot (u, v_A)$ is Earley-walk in T and $\mathit{item}(\mathcal{E}) = I$.

- $\alpha \neq \varepsilon \;\wedge\; \alpha_{|\alpha|} = B \in N$:
  This implies that I was derived by completion from $I_1 = \big(i \;\; r, A \to \alpha_{1,|\alpha|-1} \bullet B\beta\big)$ and $I_2 = \big(r \;\; j, B \to \eta \bullet\big)$ for $i \leqslant r \leqslant j$ and rule $B \to \eta$. Applying induction hypothesis to I_2 yields existence of parse tree T for $S' \Rightarrow^* w_{1,i-1}A\gamma \Rightarrow w_{1,i-1}\alpha_{1,|\alpha|-1}B\beta\gamma \Rightarrow^* w_{1,r-1}B\beta\gamma \Rightarrow^* w_{1,j-1}\beta\gamma$ exists with Earley-walk $\mathcal{E}'$ fulfilling $\mathit{item}(\mathcal{E}') = I_2$. (u, v_B) is the last edge in $\mathcal{E}'$, for u the rightmost child of v_B. For v_A the parent of v_B, we define $\mathcal{E} := \mathcal{E}' \cdot (v_B, v_A)$ and find that $\mathcal{E}$ is Earley-walk in T and $\mathit{item}(\mathcal{E}) = I$.

□

Proof of Theorem 1: Using Lemma 2, only one little step remains to be shown: If a partial derivation tree T for $S' \Rightarrow^* w_{1,i-1}A\gamma \Rightarrow w_{1,i-1}\alpha\beta\gamma \Rightarrow^* w_{1,j-1}\beta\gamma$ exists, so does an Earley-walk $\mathcal{E}$ in T with $\mathit{item}(\mathcal{E}) = \big(i \;\; j, A \to \alpha \bullet \beta\big)$.

However, this is trivial: On our way following the outline of T, we cannot meet leaves with non-terminals, since those would have to appear in the derivation above. The terminals, we find must exactly be $w_{1,j-1}$, for the same reason. So, finally we hit v_A, the node corresponding to the A in the above derivation, and find its $|\alpha|$th child. Here, we stop and have found a suitable Earley-walk. □

Efficiency of Earley parsers is heavily influenced by chosen item representations. Classical approaches use linear lists combining items with identical end index j into so-called item sets. Our approach will use a huge array storing values for all possible items, see chapter 4.

So far, we only addresses the decision problem. However, the items computed by Earley's algorithm enable a simple and efficient *traceback* to solve the parsing problem $S \stackrel{?}{\Rightarrow} w$ for CFGs as well.

CFGs are heavily used in RNA secondary structure prediction, so we next introduce the formal model for RNA.

[3] Strictly speaking, $\mathcal{E}'$ could also end with (u', v_C) if u' is the left sibling of v_A. In this case, we simply append (v_C, v_A) to $\mathcal{E}'$.

[4] Again, $\mathcal{E}'$ might end with (u', v_A) for u' the left sibling of u, in which case we append (v_A, u).

2.3. RNA

Ribonucleic acid (RNA) consists of a chain of nucleotides. This backbone of the RNA is quite stable and — once built — remains unchangedly chained in nature. Different types of nucleotides exist, depending on the nitrogenous base used. Typically, only the following four bases occur: adenine (a), cytosine (c), guanine (g) and uracil (u).

Because of interaction between different nucleotides, RNA molecules tend to *fold* three-dimensionally. We consider hydrogen bonds as possible between the base pairs $\{a, u\}$, $\{c, g\}$ and $\{g, u\}$.

We model the 'backbone', the **primary structure** of an RNA molecule, as a word $r \in \Sigma^\star$ over the alphabet $\Sigma = \{a, c, g, u\}$, written in $5'-3'$ direction. We will use lower case letters to denote primary structures.

We abstract from the actual three-dimensional folding of an RNA $r \in \Sigma^n$ by only looking at its **secondary structure** $R \subset \{1, \dots, n\}^2$, which is a set of pairs of indices satisfying

- $\forall (i,j) \in R \quad j - i \geqslant 4$
- $\forall (i,j), (k,l) \in R \quad (i = k \vee j = l) \to (i,j) = (k,l)$
- $\nexists (i,j), (k,l) \in R \quad i < k < j < l$
- The bases of all pairs $(i,j) \in R$ are either $\{a, u\}$, $\{c, g\}$ or $\{g, u\}$.

Informally speaking, that means: The pairs in R are sorted with a minimum distance of four (smaller hairpins do not form). Every base can take part in at most one pair (not in several). All pairs are *correctly nested*; non-nested base pairings are said to form a **pseudoknot** and will be excluded from secondary structures in this thesis.[5]

The first and last requirements are sometimes loosened, as hairpins of smaller size and non-canonical base pairs sometimes show up in structure databases. Two adjacent pairs (i,j) and $(i+1, j-1)$ are called **stacked**, several adjacent stacked pairs form a **stem**. Secondary structures containing many and long stems are typically considered most stable.

RNA secondary structures without pseudoknots are isomorphic to **dot-bracket** words, i. e. words over the alphabet $\{(,), |\}$ that are correctly parenthesized.[6] We use a vertical bar '|' instead of a dot for better legibility.

Dot-bracket words can be generated by CFGs. Therefore CFGs provide a model for secondary structures. But since we are not interested in all possible secondary structures — there are much too many of them — we extend our model: *Stochastic* context free grammars include a notion of "what kind of structures are more probable than others", which in turn can be used to compute a most likely secondary structure.

[5] It is known that their prediction causes problems; Lyngsø et al. show in [LP00] that prediction of such structures is $\mathcal{NP}$-complete if using an energy-minimization approach.

[6] Strict equivalence requires to exclude dot-bracket words containing any of the the words $\{(), (|), (||)\}$ as substring. Those subwords implied a hairpin of length < 3.

2.4. Stochastic context-free grammars

A **stochastic context-free grammar** (SCFG) G is a tuple $G = (N, \Sigma, R, S, P)$, such that

- (N, Σ, R, S) is a CFG and
- P is a function from R to $[0, 1]$ satisfying

$$\forall A \in N \quad \sum_{(A \to \alpha) \in R} P(A \to \alpha) = 1 \,.$$

Stated differently, for every non-terminal $A \in N$, P represents a probability distribution for R_A. A rule $A \to \alpha$ with $P(A \to \alpha) = p$ will be written shortly as $A \to p : \alpha$.

P is used to give words $w \in \mathcal{L}(G)$ a *weight* $p(w)$: The weight of w is defined as the sum of the weights of all derivation trees for w, where the weight of a derivation tree is the product of the weights for all occurrences of rules.

We would like these weights to form a probability distribution over $\mathcal{L}(G)$, but unfortunately this is not the case in general: Consider the grammar $G = (\{S\}, \{a\}, R, S, P)$ with rules $S \to \frac{2}{3} : SS$ and $S \to \frac{1}{3} : a$. At any point in a derivation, choosing the first rule is more probable, elongating the 'to-be-expanded'-list by *two* new occurrences of S, while finishing only *one* old occurrence. So it is — at any point — more probable for the list to grow than to shrink resulting in non-zero probability for infinite derivations (only using the first rule).[7] Consequently, this probability is *lost* for the finite derivations that actually generate words: $\sum_{w \in \mathcal{L}(G)} p(w) < 1$.[8] So p is *not* a probability distribution in this case.

A grammar where the $p(w)$ actually sum up to 1 is called a **consistent** SCFG. Fortunately, a nice result from [CG98] allows us to (almost) forget about inconsistency: If P is determined using **maximum likelihood estimates** from finite parses, the resulting SCFG is guaranteed to be consistent. And this is exactly, what we will do (see section 3.6 and 3.7).

2.5. SCFGs for secondary structure prediction

For secondary structure prediction, we need an ambiguous SCFG G with the following properties:

- The language of G is the set of primary structures, i. e. $\Sigma = \{a, c, g, u\}$ and $\mathcal{L}(G) = \Sigma^+$.
- Every leftmost derivation of a word $r \in \mathcal{L}(G)$ encodes a secondary structure R for r.
- For primary structure r and secondary structure R for r, there is *exactly one* derivation tree for r in G that encodes R.

So, G must ambiguously produce primary structures, but yield every possible secondary structure as exactly one possible derivation.

[7] The derivation process can be thought of as an asymmetric random walk of the number of non-terminals currently present. This number increases by one with probability $\frac{2}{3}$ and decreases by one with probability $\frac{1}{3}$. Reaching zero means finishing the derivation. But since this walk has a positive drift, it will not almost sure return to zero.

[8] See [LN10] for a formal proof of inconsistency for this grammar.

We *train* the rule probabilities P with actual data — making G a SCFG, a probabilistic model of secondary structures. Before we will see, how the training works in detail, let's have a look at prediction. If we are given a new/unknown primary structure r, we know that there are numerous derivations for r in G. Each of those encodes one possible secondary structure and is assigned a probability. So, by looking for one with maximal probability, we find the most likely secondary structure — given our model G. This derivation tree is called **Viterbi parse**.

To compute Viterbi parses, one can use adapted versions of CYK or Earley-parsers. But since the idea of semiring parsing provides a beautifully unified view on that, we leave that for the next section.

For the training of P there are always two distinct cases with respect to available data:

(1) **For all primary structures, trusted secondary structures are known.**
In this case, we construct the derivation tree corresponding to a secondary structure R — there is exactly one such tree — and count the number of occurrences for each rule. The counters are summed up for all secondary structures. Taking relative frequencies for all rules expanding one non-terminal yields a maximum likelihood estimate for P:

Because of context-*free*ness, every step in a derivation is stochastically independent of all other steps. Since the set of grammar rules is finite, the expansion of a non-terminal A is a discrete random experiment. For those it is known, that relative frequencies are a maximum likelihood estimate.[9]

(2) **Only primary structures are available.**
Even without a single secondary structure, training is possible using an evolutionary expectation maximization algorithm (see e. g. [DEKM98, chapter 9]). The idea is to start with a random/arbitrary P and iteratively improve it. In every iteration, we compute *expected* numbers of occurrences for all rules. Relative frequencies of those are then used as the next P.

For efficiently determining the expected number of occurrences of a rule, **inside and outside probabilities** are used:

$$\begin{aligned} \text{inside probability:} \quad & \alpha(A,i,j) := \Pr\left[A \Rightarrow^* w_{i,j-1}\right] \\ \text{outside probability:} \quad & \beta(A,i,j) := \Pr\left[S \Rightarrow^* w_{1,i-1}\, A\, w_{j,n}\right] \end{aligned}$$

All computations needed are slight variations of the property

$$\sum_{1 \leqslant i \leqslant j \leqslant |w|+1} 1 \cdot \alpha(A,i,j) \cdot \beta(A,i,j) = \mathbb{E}\left[\#\text{occurrences of } A \text{ in derivation for } w\right],$$

i. e. multiplying together corresponding inside and outside probabilities and summing over the possible positions.

The inside and outside probabilities themselves can be computed using adapted CFG parsers — a similar situation as for Viterbi parses. As mentioned above, semiring parsing provides a framework that embraces all of these adaptions.

[9] A more formal argumentation can be found in [CG98, section 2].

2.6. Semiring Parsing

In [Goo98, Goo99], Goodman describes parsing as a *deductive process*. The parser is specified as calculus / formal system over **items**: The parser determines the type of items used, a set of **axioms** — items to start with — and **inference rules** — rules for creating new items from existing ones. Actually, the notion of formal systems' inference rules has to be extended a little: Rules here may have **side conditions**, written as

$$\frac{A_1, \ldots, A_k}{B}\ C_1, \ldots, C_l\ .$$

The A_i are the 'real premises', C_i are side conditions, B is the conclusion. For the moment, you can think of the C_i as additional premises, i. e. for deriving B, items $A_1, \ldots, A_k$ *and* $C_1, \ldots, C_l$ must be present. (Different treatment of A_i and C_i is vital for item *values*, though. We'll come to that.) Additionally, there is a **goal item** that can be derived iff the parsing succeeds ($w \in \mathcal{L}(G)$).

Derivable items are assigned a **value** in a **semiring** $(\mathcal{U}, \oplus, \otimes, \mathbf{0}, \mathbf{1})$, which is a set $\mathcal{U}$ together with additive and multiplicative operations $\oplus$ and $\otimes$, respectively. $\mathbf{0}$ is the identity element of $\oplus$, $\mathbf{1}$ is the one[10] of $\otimes$, both operations are associative, and $\otimes$ distributes over $\oplus$. The additive operation is always commutative, $\otimes$ need not be.

The value $\nu(I)$ of an item I is determined by the deduction process: Its value is the sum over all possible rule applications with conclusion I, where each rule application contributes with the product of the premises' values, but *not* the values of the side conditions (see below). *Important*: Two rule applications differing *only* in the 'used' side conditions are counted as *one* rule application; but two applications of the same rule with *different* real premises are summed up.

Of course, all above mentioned calculations are done in the semiring. Since $\otimes$ is not necessarily commutative, the *order* of premises in a rule matters.

Goodman further describes a generic method how to 'execute' such a description of a (suitable[11]) parser to determine the values needed for using SCFGs as probabilistic models (see last subsection). We will only describe the relevant part of it here; in particular we will **exclude** grammars with **loops**.[12]

First, a total order $\prec$ on all (derivable) items is determined, such that in all possible rule applications with premises $A_1, \ldots, A_k$, side conditions $C_1, \ldots, C_l$ and conclusion B $A_1, \ldots, A_k, C_1, \ldots, C_l \prec B$ holds.

Then, the items are processed in order $\prec$ and for every item B, its value is computed:

$$\nu(B) = \bigoplus_{\substack{A_1, \ldots, A_k : \\ \frac{A_1, \ldots, A_k}{B}\ C_1, \ldots, C_l}} \nu(A_1) \otimes \cdots \otimes \nu(A_k)\ .$$

[10] no pun intended

[11] Not every set of deduction rules forms a correct parser capable of computing semiring values. Since the Earley-style parser we will use is essentially the one Goodman proposes in [Goo98], we will ignore this 'subtlety'.

[12] Such grammars immediately allow an infinite number of derivation trees for one word. This does never make sense in the structure-prediction context since the number of possible structures for one object is always finite. The delightful consequence is that we will never have looping buckets, so we do not even need the concept of buckets: It will always be possible to strictly sort the items.

All items A which $\nu(B)$ might depend on have $A \prec B$, hence their values have already been computed.[13] If the goal item is reached that way, the parse was successful.

In this setting, VITERBI parses and inside values only require to use the correct semiring, namely:

- The **Viterbi-semiring** $\big([0,1], \max, \cdot, 0, 1\big)$ computes the probability of the most likely derivation tree for the given input.[14]
- The **inside-semiring** $\big(\mathbb{R}_{\geqslant 0} \cup \{\infty\}, +, \cdot, 0, 1\big)$ is the most 'natural' instantiation of the semiring because $\oplus \equiv +$ and $\otimes \equiv \cdot$ on non-negative real numbers. Including the real numbers > 1 and ∞ is due to 'mathematical technicalities': Formally, the semirings have to be closed under infinite summation. The kind of calculations actually performed in our parser are (a) always finite (no loops!) and (b) guaranteed to never exceed 1. So, no problem with that.

Outside probabilities are a bit more problematic, but GOODMAN offers a solution to that, as well: For commutative semirings, **reverse values** — a generalization of the connection between outside and inside probabilities — can be computed by iterating over the items in 'reverse $\prec$ order' (i. e. $\succ$) and using a slightly different formula, namely: The reverse value $z(A)$ of item A is $\mathbf{1}$, if A is the goal item and otherwise determined by

$$z(A) = \bigoplus_{\substack{A_1,\dots,A_k,j,B\,:\\ \frac{A_1,\dots,A_k}{B}\ C_1,\dots,C_l \ \wedge\ A_j=A}} z(B) \otimes \nu(A_1) \otimes \cdots \otimes \nu(A_{j-1}) \otimes \nu(A_{j+1}) \otimes \cdots \otimes \nu(A_k)\,.$$

Notice that for computing the reverse values, the forward values must already be known. Reversing the direction of iteration ensures, that the reverse value $z(B)$ in the formula above is known, when computing $z(A)$.

Finally, the reverse values in the inside-semiring correspond to outside probabilities. This is all we need to train a probabilistic model based on SCFGs and to use it for prediction. In fact, inside and outside probabilities turned out to have uses beyond expectation maximization; see section 6.1 on page 80.

The idea of semiring parsing allows us to concentrate on the actual parsing technique by automating some of the gory details of probabilistic parsers. As our actual goal is to go beyond context-free grammars — where parsing itself gets more complex — this help is very welcome. But before we introduce the new grammar types, let's look at the things we will want to predict.

[13] $A \prec B$ has the meaning of $A \precneqq B$, i. e. an item B's value will never depend on its *own* value. This awkward situation may happen in grammars with loops, which we excluded from our discussion (see footnote 12).

[14] We actually need more! For structure prediction we need to have the whole VITERBI parse, i. e. the derivation tree. GOODMAN also presents a special semiring that directly keeps track of the tree while parsing; but this semiring is not commutative, which causes some complications we prefer to avoid. Therefore we will simply compute all item values in the VITERBI-semiring and do a backtrace step afterwards that reconstructs the actual parse tree.

2.7. RNA-RNA Interaction

It has been observed that for understanding the biological function of RNA molecules, it is not sufficient to look at the molecule and its secondary structure in isolation. One important aspect overlooked that way is the *interaction* of RNA molecules. During protein biosynthesis such interactions can hinder mRNA leaving the nucleus, thereby controlling gene expression.

We will model these effects by looking at *two* RNA molecules with primary structures $r \in \Sigma^n$ of length n and $s \in \Sigma^m$ of length m, respectively; as before $\Sigma = \{a, c, g, u\}$. Both r and s are given in $5'-3'$ direction.

As with single RNAs, we abstract from the three-dimensional folding and rather look at the **joint (secondary) structure** $\binom{R}{S}$, which is a set of pairs of indices in r and s. Regarding the indices, we will think of the *concatenation* $r \cdot s$, i. e. the indices in s start at $n+1$ and end at $n+m$. Then we can formally define the joint structure as $\binom{R}{S} \subset \{1, \dots, n+m\} \times \{1, \dots, n+m\}$ satisfying some conditions — those conditions are easier to state using the following subsets of $\binom{R}{S}$:

$$\begin{aligned} R &:= \left\{(i,j) \in \tbinom{R}{S} : i \leqslant n \wedge j \leqslant n\right\}, \\ S &:= \left\{(i,j) \in \tbinom{R}{S} : i > n \wedge j > n\right\}, \\ \text{////} &:= \tbinom{R}{S} \setminus (R \cup S)\,. \end{aligned}$$

Those subsets are called the sets of **internal base pairs** for r and s and the set of **external base pairs**, respectively. Now we can give the conditions $\binom{R}{S}$ must fulfill as a valid joint structure:

- No hairpins shorter than 4:
  $\forall (i,j) \in R \quad j - i \geqslant 4$ and
  $\forall (i,j) \in S \quad j - i \geqslant 4$.
- Only disjoint pairs:
  $\forall (i,j), (k,l) \in \binom{R}{S} \quad (i = k \vee j = l) \to (i,j) = (k,l)$.

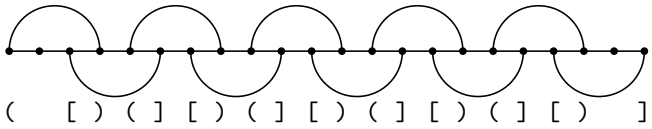
We excluded pseudoknots from secondary structures, since their prediction is much more difficult. The infeasibility results directly carry over to joint structures, so we will want to forbid pseudoknots here, as well. But since we now have different kinds of pairings — namely internal and external base pairs — we also have different types of pseudoknots.

Two of them are straight-forward; the typical characterization of pseudoknots applied to our subsets of $\binom{R}{S}$: We call a joint structure $\binom{R}{S}$

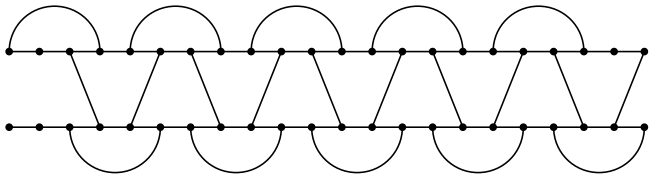
- free of **internal pseudoknots**, if
  $\nexists (i,j), (k,l) \in R \quad i < k < j < l$ and
  $\nexists (i,j), (k,l) \in S \quad i < k < j < l$.
- free of **external pseudoknots**, if
  $\nexists (i,j), (k,l) \in \text{////} \quad i < k < j < l$.

Digging a little deeper into what makes conventional (one RNA) pseudoknots hard to predict, we find that efficient methods require *disjoint partitioning* of the primary structure.

In case of pseudoknot-free structures, so-called k-loops provide such partitioning (e. g. see [ZS84]). This does not work with pseudoknots, as they can — in a sense — grow beyond any bounds with 'increasing complexity':[15]



Of course, this pseudoknot can be extended arbitrarily; trivial attempts to partition it — e. g. vertical slicing — do not yield disjoint substructures. This makes it hard to predict. A similar situation can occur in joint structures, even if we do not allow the above mentioned pseudoknots — due to the similarity you might call it a **'mixed pseudoknot'**:



So, one expects these kinds of joint structures to cause problems, too, and indeed they do. In [AKN+06], Alkan et al. show that even the simple task of maximizing the number of base pairs is $\mathcal{NP}$-complete for joint structures, if such mixed pseudoknots are allowed.

Since our approach will be based on a disjoint partitioning scheme, as well — namely the one proposed in [HQRS10] — we have to set up another constraint, which forbids situations as in the picture above: A joint structure $\binom{R}{S}$ is called

- free of **zig-zags/mixed pseudoknots**, if
  for all $(i,j) \in R$ and $(k,l) \in S$ one of the following is fulfilled
  - $\nexists (p,q) \in \binom{R}{S} \quad i < p < j \ \wedge \ k < q < l$ or
  - $\forall (p,q) \in \binom{R}{S} \quad i < p < j \ \rightarrow \ k < q < l$ or
  - $\forall (p,q) \in \binom{R}{S} \quad i < p < j \ \leftarrow \ k < q < l$.

This means, either there are *no* external bonds 'connecting' the internal bonds (i,j) and (k,l) — or — one of the internal pairs has to 'subsume' the other.

In the following, we will restrict ourselves to joint structures without pseudoknots and zig-zags: The problem "given r and s, determine such $\binom{R}{S}$", is called the **RNA-RNA interaction problem** (RIP).

Context-free grammars are not suited for modelling joint structures as the two kinds of pairs — internal and external bonds — are allowed to *cross* in a non-nested fashion. There is, however, a nice generalization concept that will result in a grammar class powerful enough to handle joint structures: *multiple* context-free grammars.

[15] Admittedly, this argumentation is wishy-washy. The unsatisfied reader may regard it as a mere motivation for the definition of zig-zags, that will immediately follow the paragraph with the attractive pictures.

2.8. Multiple context-free grammars

Applications in computational biology — especially RIP and pseudoknots prediction — need more expressiveness than CFGs can offer. On the other hand, efficient parsing must be maintained for practical applicability. Therefore, advances in the field have yielded a zoo of formal grammar classes fulfilling these requirements, commonly referred to as **weakly context-sensitive grammars**. Many of those turned out to be special cases of so-called multiple context-free grammars (see [Kato7] for example), so it seems worth having a look at these.

Our definition of multiple context-free languages differs a little from typical introductions. This is a try to overcome some notational monstrosities, that arise from defining MCFGs as special case of the even more general class of general context-free grammars. For a thoroughly formal introduction and a collection of interesting properties, see [SMFK91].

A **multiple context-free grammar** (MCFG) is a tuple $G = (N, d, \Sigma, R, S)$ comprised of

- N, a finite set of **non-terminals**,
- d, a function from N to $\mathbb{N}$, assigning a **dimension** $d(A)$ to each $A \in N$,
- Σ, a finite set of **terminals**,
- R, a finite set of **rules** and
- the **start symbol** $S \in N$ with $d(S) = 1$.

In MCFGs, non-terminals $A \in N$, may produce **tuples** or **vectors** of words whose dimension is given by $d(A)$. For a non-terminal $A \in N$ and an index $i \in \{1, \dots, d(A)\}$, we write A_i for the ith component of A. Restricting $d(A) = 1$ for all non-terminals A yields a plain CFG (which explains the name M**CFG**).

The most interesting part of the definition of MCFGs is still missing yet: How do multi-dimensional rules look like? The general form is again quite similar to CFGs, just that it is in terms of 'vectors' instead of 'scalars':

$$\text{short vector form:} \qquad \begin{array}{c} A \to \alpha\,, \\ \text{(or more explicitly: } \vec{A} \to \vec{\alpha}\text{)} \end{array}$$

$$\text{fully expanded form:} \qquad \begin{pmatrix} A_1 \\ \vdots \\ A_{d(A)} \end{pmatrix} \to \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_{d(A)} \end{pmatrix}.$$

The α_i are words over $\Sigma \cup \mathcal{C}$, where $\mathcal{C} := \{A_i : A \in N \wedge i \in \{1, \dots, d(A)\}\}$, the set of **non-terminal components**. In every rule, each non-terminal component may be used *at most once* in the α_i *in total*, i. e. it may appear at most once in the word $\alpha_1 \alpha_2 \cdots \alpha_{d(A)}$.[16] Although the definition is quite simple, stressing some aspects might improve understanding.

[16] Using this approach forbids usage of one non-terminal several times in a right hand side, as in $A \to BB$. However, typical definitions of MCFGs allow such rules. By simply introducing a new non-terminal C and replacing $A \to BB$ by $A \to BC$ and $C \to B$, a strictly equivalent grammar fulfilling the restriction can be created. Alternatively, one might change our definition to include a notation for multiple usage of one non-terminal.

- We do not restrict the order of the non-terminal components in the α_i. So B_2 might jump in front of B_1. Actually, this is a vital feature of MCFGs.
- The α_i may contain arbitrary terminal strings, surrounding and separating the non-terminal components. If no non-terminal components appear at all, such an $A \to \alpha$ is called a **terminating rule**.
- The definition allows one component of a non-terminal to be used, even if no / not all other components (of that very same non-terminal) appear.

  These kinds of deletions are often *not* wanted — as they complicate parsing — and are in fact *not necessary* (see Lemma 2.2 of [SMFK91]). Therefore, we will in the following assume that each non-terminal appears in the right hand side of a rule either
  - ▷ completely and once — i. e. each component exactly once — or
  - ▷ not at all.

In order to define derivations in a MCFG, we need the term of **non-terminal incarnations**: For $A \in N$ and $s \in \mathbb{N}$ we call $A^{\langle s\rangle}$ the sth incarnation of A. We will also write $A_i^{\langle s\rangle}$ for the ith component of $A^{\langle s\rangle}$ and call $\mathcal{I} := \left\{A_i^{\langle s\rangle} : A \in N \wedge s \in \mathbb{N} \wedge i \in \{1, \dots, d(A)\}\right\}$ the set of all non-terminal incarnation components. For convenient usage, we extend this notation to arbitrary words $\gamma \in (\Sigma \cup \mathcal{C})^\star$ of terminals and non-terminal components: $\gamma^{\langle s\rangle} \in (\Sigma \cup \mathcal{I})^\star$ is the word resulting from γ, when all components A_i are replaced by incarnations $A_i^{\langle s\rangle}$.

We will use this 'incarnation marker' s to able to tell which non-terminal components belong together — only those can be expanded by rules. We define the **left-most derivation** relation $\underset{G}{\Rightarrow} \subseteq (\Sigma \cup \mathcal{I})^\star \times (\Sigma \cup \mathcal{I})^\star$ by

$$x\ A_{i_1}^{\langle s\rangle}\beta_1\ A_{i_2}^{\langle s\rangle}\beta_2\ \cdots\ A_{i_{d(A)}}^{\langle s\rangle}\beta_{d(A)} \quad \underset{G}{\Rightarrow} \quad x\ \alpha_{i_1}^{\langle t\rangle}\beta_1\ \alpha_{i_2}^{\langle t\rangle}\beta_2\ \cdots\ \alpha_{i_{d(A)}}^{\langle t\rangle}\beta_{d(A)}$$

$$\begin{aligned}
\text{iff} \quad & x \in \Sigma^\star \\
\wedge \quad & A \to \alpha \in R \\
\wedge \quad & \{i_1, \dots, i_{d(A)}\} = \{1, \dots, d(A)\} \\
\wedge \quad & s \in \mathbb{N} \\
\wedge \quad & t > \text{any incarnation number in } \beta_1\beta_2\cdots\beta_{d(A)}\,.
\end{aligned}$$

So, left-most derivation in MCFGs means expanding the non-terminal whom the leftmost component belongs to. Since components may appear shuffled on the left, we have to re-index them. Again we define $\Rightarrow^*$ to be the reflexive and transitive closure of $\Rightarrow$ and are — finally — able to define the **language** of a MCFG G:

$$\mathcal{L}(G) := \left\{w \in \Sigma^\star : S^{\langle 0\rangle} \underset{G}{\Rightarrow}^* w\right\}.$$

Two things are left to discuss about $\underset{G}{\Rightarrow}$:

(1) The condition for t might seem problematic; how to choose it?
Fortunately, for complete derivations — i. e. ones starting at $S^{\langle 0 \rangle}$ and producing a terminal word — setting t to the *number* of performed *rule applications* suffices.[17]

(2) What about (partial) derivations of *multi-dimensional* non-terminals?
Indeed, our definition of $\underset{G}{\Rightarrow}$ can only handle 'flat' strings, but this is no serious problem[18]. Let \$ be a symbol not appearing in $N \dot\cup \Sigma$. Then we have a one-to-one mapping between the vectors $(\gamma_1, \gamma_2, \ldots, \gamma_d)$ of strings $\gamma_i \in (\Sigma \cup \mathcal{J})^\star$ for all $i \in \{1, \ldots, d\}$ and the 'flattened' strings $\gamma_1 \$ \gamma_2 \$ \cdots \$ \gamma_d$. Now we can safely let $\underset{G}{\Rightarrow}$ operate on the flattened string, because it will treat \$ as a terminal, i. e. it will simply ignore it. Afterwards we transform the new flat string back to the tuple.[19]

2.9. Stochastic multiple context-free grammars

For serving as probabilistic model, we have to extend MCFGs with rule probabilities — knowing how to do that with CFGs, this is easy:

A **stochastic multiple context-free grammar** (SMCFG)[20] G is a tuple $G = (N, d, \Sigma, R, S, P)$, such that

- (N, d, Σ, R, S) is a MCFG and
- P is a function from R to $[0, 1]$ satisfying $\forall A \in N \sum_{(A \to \alpha) \in R} P(A \to \alpha) = 1$.

As we did for SCFGs, we will set the *weight* of a derivation to the product of the rule probabilities and the weight of a terminal word to the sum of its derivations. The consistency results for SCFGs carry over, as well: Both in SCFGs and in SMCFGs, the choice of a rule to expand some non-terminal A works exactly the same way: It is a discrete random experiment and all expansions of A are independantly and identically distributed. Therefore, the proof in [CG98] works for SMCFGs as well. Consequently, we will refer to the weights as probabilities.

For the training of P with unknown structures, we need generalized inside and outside probabilities. As stressed in the definition, rules of MCFGs can produce the components of a multi-dimensional non-terminal A in *any order*. Of course, the order matters for continuing the derivation. This fact makes the definition of inside and outside probabilities more complicated than in the SCFG-case. We need to include information about the *order* of the non-terminal-components: Let $\{i_1, \ldots, i_{d(A)}\} = \{1, \ldots, d(A)\}$ be a permutation of the components. Furthermore, we have *ordered* indices in w

$$1 \leqslant l_1 \leqslant r_1 \leqslant \cdots \leqslant l_{d(A)} \leqslant r_{d(A)} \leqslant |w| + 1 \,.$$

[17] Formalizing this further, one can include this number of rule applications in the elements $\underset{G}{\Rightarrow}$ is defined on, making the definition more 'self-contained'. For the sake of clarity and brevity, we will not do that.

[18] … seriously!

[19] Alternatively, one can again fiddle with the definition of $\underset{G}{\Rightarrow}$ to directly work on tuples. That would — at least — introduce another index and complicate the term left-most. I know which version I prefer.

[20] We will not go beyond five-letter acronyms … I promise!

We define

$$\alpha(A\,;\, i_1,\dots i_{d(A)}\,;\, l_1,r_1,\dots,l_{d(A)},r_{d(A)}) \coloneqq \Pr\left[\begin{pmatrix} A_1 \\ \vdots \\ A_{d(A)} \end{pmatrix} \Rightarrow^* \begin{pmatrix} w_{l_{i_1},r_{i_1}-1} \\ \vdots \\ w_{l_{i_{d(A)}},r_{i_{d(A)}}-1} \end{pmatrix}\right],$$

$$\beta(A\,;\, i_1,\dots i_{d(A)}\,;\, l_1,r_1,\dots,l_{d(A)},r_{d(A)}) \coloneqq \Pr\left[S \Rightarrow^* w_{1,l_1-1}\ A_{i_1}\ w_{r_1,l_2-1}\ A_{i_2}\ w_{r_2,l_3-1}\ \cdots\ w_{r_{d(A)-1},l_{d(A)}-1}\ A_{i_{d(A)}}\ w_{r_{d(A)},|w|}\right].$$

Of course, for most grammars, many of those probabilities will be zero, e. g. because a given permutation of non-terminal components cannot be derived at all — but this is perfectly OK from a theoretical point of view.

correcting sloppiness

Strictly speaking, the above probabilities are not well-defined: Our derivation relation $\Rightarrow$ only works on non-terminal *incarnations*. Yet, repair to the definitions is possible: For the inside probabilities, we simply replace A_i by $A_i^{\langle s\rangle}$ for an arbitrary $s \in \mathbb{N}$. This suffices because $\Rightarrow$ is defined to operate on any incarnation number and there are infinitely many natural numbers greater than any $s \in \mathbb{N}$. So the inside probability does not depend on s.

In the definition of the outside probabilities, we first replace S by $S^{\langle 0\rangle}$. For the A_i, we run into a little problem: Any incarnation number s for the A_i might restrict the number of derivation steps allowed from $S^{\langle 0\rangle}$! But there may very well be *infinitely* many derivations of growing length, all contributing essentially to the outside probability. This means, any finite bound on the number of allowed derivation steps leads to truncation.

Note that this does neither mean that we have to deal with infinitely *long* derivations, nor with non-terminal incarnations '$A^{\langle\infty\rangle}$': In the end, we are only interested in *terminating* derivations, i. e. derivations producing a (finite) terminal string. Those derivations are always finite themselves as there must be a *last* derivation step that eliminates the last non-terminal left. We actually do allow infinitely *many* derivations to contribute to both inside and outside probabilities; therefore we could not give any fixed bound on s in the definition of outside probability. Yet, all those derivations have finite length — infinite derivations must have probability zero in a consistent grammar.

So we have to take the *limit* of the stated probability as s goes to infinity:

$$\beta(A\,;\, i_1,\dots i_{d(A)}\,;\, l_1,r_1,\dots,l_{d(A)},r_{d(A)}) \coloneqq \lim_{s\to\infty} \Pr\left[S^{\langle 0\rangle} \Rightarrow^* w_{1,l_1-1}\ A_{i_1}^{\langle s\rangle}\ \cdots\ w_{r_{d(A)-1},l_{d(A)}-1}\ A_{i_{d(A)}}^{\langle s\rangle}\ w_{r_{d(A)}+1,|w|}\right].$$

This limit has to exist as the underlying sequence is monotonically increasing and bounded above by 1.

2.10. Parsing of multiple context-free grammars

Using SMCFGs as fully training-enabled models for joint structures, we need a semiring parser for SMCFGs.

In [SMFK91] it is shown that the *decision problem* '$w \overset{?}{\in} \mathcal{L}(G)$' for general MCFG $G = (N, d, \Sigma, R, S, P)$ can be solved in $\mathcal{O}(|w|^e)$, where e is the **degree** of G:

$$e = \max_{A \to \alpha \in R} \left\{ d(A) + \sum_{\text{non-terminal } B \text{ in } \alpha} d(B) \right\}.$$

Remember that according to our definition of MCFGs, a non-terminal B is not allowed to occur several times in the right hand side of a rule. In the expanded rule form, e is simply the overall number of non-terminal *components* — including the ones *left* of '$\to$'!

As an example, we consider the grammar G_{RIP} we will use later — see Grammar 1 on page 39. The rule $E \to KFD$ is actually an abbreviation for $\binom{E_1}{E_2} \to \binom{K_1 F_1 D_1}{K_2 F_2 D_2}$ — you may think of E, K, F and D as 2D-vectors. This rule therefore has *eight* non-terminal components in total. No other rule has more than eight such components, hence we have $e = 8$ for G_{RIP}. So we know we can decide $\mathcal{L}(G_{\text{RIP}})$ in $\mathcal{O}(|w|^8)$.

The algorithm of Seki et al. is conceptionally similar to a CFG chart parser, so using it as semiring parser is possible *in principle*. There have also been approaches to explicitly state a general MCFG Earley-parser in [Alb02], more directly implementable as semiring parser, but with worse running time complexity.

The general parsing algorithm in [SMFK91] — and the same holds for all general MCFG parsers known to the author — is quite complex and hence hard to implement. Moreover it has

- large constant factors hidden in the $\mathcal{O}$-notation for runtime and
- restrictively high memory consumption — at least without further non-trivial optimizations.

But fortunately, for tackling the RNA-RNA interaction problem, full expressive power of MCFGs is not needed and more efficient parsing is possible.

3. Approach

3.1. m-dimensional context-free grammars

The grammar we will use does, by far, not exhibit all the features and complications that general MCFGs may have. So, we will confine our parser to a subclass of (S)MCFGs containing our grammar that allows much easier parsing.[21]

A **(stochastic) m-dimensional context-free grammar** (mD-CFG[22]) is a tuple $G = (N, \Sigma, R, S, P)$ such that

- $\overline{G} := \left(N \dot\cup \{S'\},\ d,\ \Sigma,\ R \dot\cup \{S' \to S_1 S_2 \cdots S_m\},\ S',\ P'\right)$ with

  $$d(A) := \begin{cases} 1 & \text{if } A = S' \\ m & \text{otherwise} \end{cases} \qquad \text{and} \qquad P'(r) := \begin{cases} 1 & \text{if } r = S' \to S_1 S_2 \cdots S_m \\ P(r) & \text{otherwise} \end{cases}$$

  is a SMCFG.

- For all rules $A \to \gamma \in R$ holds for some (rule-dependent) $L \in \mathbb{N}_0$

  $$\begin{pmatrix} \gamma_1 \\ \vdots \\ \gamma_m \end{pmatrix} = \gamma = x^{(1)}\, B^{(1)}\, x^{(2)}\, B^{(2)} \cdots x^{(L)}\, B^{(L)} = \begin{pmatrix} x_1^{(1)}\, B_1^{(1)} & \cdots & x_1^{(L)}\, B_1^{(L)} \\ & \vdots & \\ x_m^{(1)}\, B_m^{(1)} & \cdots & x_m^{(L)}\, B_m^{(L)} \end{pmatrix},$$

  where $x^{(i)} \in (\Sigma^\star)^m$ and $B^{(i)} \in N$ for all $1 \leqslant i \leqslant L$.

The first condition implies in particular, that *all* non-terminals have dimension m, but this is rather formal convenience, as section 3.1.2 on the following page will show.

However, the second requirement is the one implementing really influential restrictions: It forces all non-terminal components to

(a) stick with their dimension — a component A_i will only appear in dimension i — and

(b) forbids *crossings* of components from several non-terminals — e. g. we are not allowed to have a rule $A \to \binom{B_1 C_1}{C_2 B_2}$. Especially these crossings make parsing of MCFGs complicated — we will see that with the additional restrictions of mD-CFGs, parsing can be generalized from plain old CFGs rather easily.

[21] Strictly speaking, mD-CFGs are *not* a subclass of SMCFGs — only the extended version $\overline{G}$ of an mD-CFG G *is a* SMCFG. We will stick to this inaccuracy for ease of writing. The more formally inclined reader may imagine the class of grammars $\overline{G}$, such that G is an mD-CFG — a proper subclass of SMCFGs which all important results directly carry over to.

[22] Better may be mD-**S**CFG — but I keep my promises and obey the NMTFLA rule: **N**o **m**ore-**t**han-**f**ive-**l**etter-**a**cronyms. Actually, we will never need mD-CFGs that are *not* stochastic, so it is OK to have no name for those.

Two further implications of the rule restriction shall be mentioned explicitly to be taken note of:

- ε-rules are allowed, as $L = \emptyset$ is just fine.
- Since $\overline{G}$ is a SMCFG, the $B^{(i)}$ on the right hand side of one rule have to be *distinct*. However, forbidding crossings renders this restriction redundant: We are able to tell which non-terminal components belong together at any time — by counting the number of non-terminal components left to it (in the same dimension). The components where these counts agree belong together.

  So, we will **re-allow repeated use** of a non-terminal in the following.[23]

3.1.1. Derivations and language

As $\overline{G}$ is a SMCFG, we inherit a lot of concepts from SMCFGs for mD-CFG G. However, most of them are overly complicated given the restrictions of mD-CFGs. Therefore, we utilize our additional knowledge for some simplifications.

Given that crossings of non-terminal components are not allowed, every derivable sentential form has a unique leftmost non-terminal — whose components are the leftmost ones of their kind *in every component* of the sentential form. This means, that we always know which non-terminal components belong together. So, there is no need for the concept of *incarnations* — we simply define the leftmost derivation relation $\underset{\overline{G}}{\Rightarrow} \subseteq \left(N \cup (\Sigma^\star)^m\right)^\star \times \left(N \cup (\Sigma^\star)^m\right)^\star$ in terms of m-tuples by

$$x A \alpha \underset{\overline{G}}{\Rightarrow} x \gamma \alpha \qquad \text{iff} \qquad A \to \gamma \in R\,,$$

for $x \in (\Sigma^\star)^m$. Closures of $\underset{\overline{G}}{\Rightarrow}$ are defined as always. Now the language of an mD-CFG is

$$\mathcal{L}(G) := \left\{ w \in (\Sigma^\star)^m : S \underset{\overline{G}}{\Rightarrow}^* w \right\} \subseteq (\Sigma^\star)^m\,,$$

consisting of terminal m-*tuples* rather than flat strings.

3.1.2. Effective dimension

Requiring all non-terminals to have dimension m keeps the components of derived tuples 'in sync', i. e. the rules applied always concern every dimension. However, this is no limitation. If we need parts of our tuples to be derived 'non-coupled', i. e. the derivation in one dimension is independent from the derivation of another, we simply model this by two non-terminals — each solely operating in *some* dimensions.

We formalize this idea a little bit. For a non-terminal $A \in N$, we call

$$D_{\neq\varepsilon}(A) := \left\{ i : \exists x \in (\Sigma^\star)^m \;\; A \Rightarrow^* x \wedge x_i \neq \varepsilon \right\} \subseteq \{1, \ldots, m\} \text{ and}$$
$$d_{\neq\varepsilon}(A) := \left| D_{\neq\varepsilon}(A) \right|$$

the **effective indices** and **effective dimension**, respectively.

[23] To stay conform with our definition, we eliminate double uses of one non-terminal by introducing new non-terminals when constructing $\overline{G}$. See also footnote 16 on page 24.

The effective indices are all dimensions, where the non-terminal can derive a terminal string other than the empty string.

Two non-terminals $A, B \in N$ with $D_{\neq\varepsilon}(A) \cap D_{\neq\varepsilon}(B) = \varnothing$ can be concatenated to start two independent derivation parts for the components $D_{\neq\varepsilon}(A)$ and $D_{\neq\varepsilon}(B)$. If $D_{\neq\varepsilon}(A) = \{1, \dots, k\}$ and $D_{\neq\varepsilon}(B) = \{k+1, \dots, m\}$, we will write $\binom{A}{B}$ instead of AB for convenience.[24]

3.1.3. Subgrammars

As parsing complexity depends on the dimension of the grammar, the number of non-terminals and rules, we can improve efficiency by splitting off parts with *less* (effective) dimension than the whole grammar. We give some formal definitions here providing a theoretical basis to this optimization.

For the rest of this section, $G = (N, \Sigma, R, S, P)$ is an mD-CFG. First, we define the directed **reachability graph** $\mathcal{N}(G)$ of non-terminals by

$$\mathcal{N}(G) := \Big(N\ ,\ \big\{(A, B) : A \to \alpha B \beta\ \in R\big\}\Big)\ .$$

We immediately obtain: B is *reachable* from A in $\mathcal{N}(G)$, iff $A \Rightarrow^+ \alpha B \beta$. Next, we observe a nice property of $D_{\neq\varepsilon}$:

$$A \Rightarrow^* \alpha B \beta \qquad \text{implies} \qquad D_{\neq\varepsilon}(B) \subseteq D_{\neq\varepsilon}(A)\ . \tag{3.1}$$

If that were not the case, i. e. if B had a dimension $j \in D_{\neq\varepsilon}(B) \setminus D_{\neq\varepsilon}(A)$, where it can derive a terminal tuple x with $x_j \neq \varepsilon$, then we immediately have $A \Rightarrow^* \alpha x \beta =: x'$ with $x'_j \neq \varepsilon$. ↯

We can interpret (3.1) with respect to $\mathcal{N}(G)$: Assume we partition N into equivalence classes according to $D_{\neq\varepsilon}$. Then, the graph on these equivalence classes is *acyclic*. Stated differently, all cycles in $\mathcal{N}(G)$ stay within one equivalence class.

Now, we define G_A, the **subgrammar of G induced by A**, for non-terminal $A \in N$ by

$$\begin{aligned} G_A &:= \big(N_A\ ,\ \Sigma\ ,\ \textstyle\bigcup_{B \in N_A} R_B\ ,\ A\ ,\ P\big)\ , \\ \text{with} \qquad N_A &:= \big\{A\big\} \cup \big\{B : B \text{ reachable from } A \text{ in } \mathcal{N}(G)\big\}\ . \end{aligned}$$

(Remember: R_B is the set of all rules expanding B.) G_A is a $d_{\neq\varepsilon}(A)$-dimensional context-free grammar because of property (3.1).[25]

Let us call $N_m := \big\{M \in N : d_{\neq\varepsilon}(M) = m\big\}$, the set of all non-terminals of 'full' dimension. For a given mD-CFG G, we can now identify the set $\mathcal{A}$ of all non-terminals A satisfying both of the following:

[24] You might wonder "why not simply allow non-terminals of dimension $< m$ right from the start?" Assume we did that and consider a 4D-CFG containing the 2-dimensional non-terminal A. If we do not restrict the dimensions A may appear in, we might get the sentential form (A_1, A_1, A_2, A_2) … now which components belong together? To disallow such situations, you may restrict A's dimensions … which is actually equivalent to our approach! So, we had better stick to it.
"But wait a second, we just allowed to write $\binom{A}{B}$, can't the same problem occur there, as well, if $A = B$!?" Fortunately, no. We only allowed to write $\binom{A}{B}$ if $D_{\neq\varepsilon}(A) \cap D_{\neq\varepsilon}(B) = \varnothing$, so unless $d_{\neq\varepsilon}(A) = d_{\neq\varepsilon}(B) = 0$ — in which case we simply delete A and B, as they can *only* derive ε^m — we must have $A \neq B$. And then, of course, A_1 belongs to A_2 and B_1 to B_2.

[25] Formally, all non-terminals still have dimension m, as do the rules. Property (3.1), however, tells us that all dimensions $j \notin D_{\neq\varepsilon}(A)$ have to derive ε, so we can simply ignore those.

- $d_{\neq\varepsilon}(A) < m$
- There is a rule $M \to \alpha A \beta$ for an $M \in N_m$.

For all those non-terminals $A \in \mathcal{A}$, we create their subgrammars G_A and *'delete'* them from G — leaving a purely m-dimensional grammar $G_{-\mathcal{A}}$

$$G_{-\mathcal{A}} := \left(N_m \cup \mathcal{A}\ ,\ \Sigma\ ,\ \bigcup_{M\in N_m} R_M\ ,\ S\ ,\ P\right).$$

Note that $G_{-\mathcal{A}}$ is not a sensible grammar in isolation, as non-terminals in $\mathcal{A}$ remain unexpanded. Instead $G_{-\mathcal{A}}$ has to be used in conjunction with the G_A: We start derivations in $G_{-\mathcal{A}}$ and whenever we reach an $A \in \mathcal{A}$ in a derivation, we switch to G_A and continue the derivation there.

For semiring parsing of a terminal tuple $w \in \Sigma^{n_1} \times \cdots \times \Sigma^{n_m} \subset (\Sigma^\star)^m$ we will do the inverse thing: First, for each $A \in \mathcal{A}$ we parse *all substrings* of the appropriate dimensions of w according to G_A, i. e. if $D_{\neq\varepsilon}(A) = \{i_1, \ldots, i_d\}$ with $1 \leqslant i_1 < \cdots < i_d \leqslant m$, we parse

$$\begin{pmatrix} (w_{i_1})_{l_1,r_1} \\ \vdots \\ (w_{i_d})_{l_d,r_d} \end{pmatrix} \qquad \text{for all} \qquad \begin{matrix} 1 \leqslant l_1 \leqslant r_1 \leqslant n_{i_1} \\ \vdots \\ 1 \leqslant l_d \leqslant r_d \leqslant n_{i_d} \end{matrix}$$

and store the result of the parse in $v(A; l_1, r_1, \ldots, l_d, r_d)$ — the forward value of an item deriving the given terminal tuple from A. These values can then be used by a parser for $G_{-\mathcal{A}}$ to directly step over non-terminals in $\mathcal{A}$. Details on this strategy for parsing will be discussed in section 3.5 for our Earley-style parser.

3.1.4. Rule Templates

Let $A \to \gamma^{(1)}, A \to \gamma^{(2)}, \ldots, A \to \gamma^{(c)}$ be rules of an mD-CFG $G = (N, \Sigma, R, S, P)$ with the same left hand side A. Further assume, the right hand sides satisfy $f(\gamma^{(1)}) = f(\gamma^{(2)}) = \cdots = f(\gamma^{(c)})$ for homomorphism f defined by

$$f(x) = \begin{cases} x & \text{if } x \text{ is a non-terminal component} \\ t & \text{if } x \text{ is a terminal} \end{cases},$$

for a *placeholder symbol* $t \notin N \cup \Sigma$ i. e. the $\gamma^{(i)}$ are equal except for terminals. Note that for such rules, the right hand sides have the same total length and contain the same number of terminals, say e.

Let γ be the word that results from $f(\gamma^{(1)})$ if the t's are replaced by the *distinct* placeholders $t_1, \ldots, t_e \notin N \cup \Sigma$. So γ is a word (-tuple) containing non-terminal components and the t_i-placeholders. We call $A \to \gamma$ a **rule template of order e** for the rules $A \to \gamma^{(1)}, \ldots, A \to \gamma^{(c)}$ with **transition probability**[26] $P(A \to \gamma) := \sum_{i=1}^{c} P(A \to \gamma^{(i)})$. The transition probability $P(A \to \gamma)$ therefore is the probability that — given an occurrence of non-terminal A — the rule template $A \to \gamma$ is selected from the set of all rule templates expanding A. The rules $A \to \gamma^{(i)}$ are called **instances** of the template.

[26] The terminology stems from hidden Markov models, where the *transition* into a state and the *emission* of a terminal symbol are considered two separate steps.

Now, let $\gamma(a_1,\dots,a_e)$ for $a_1,\dots,a_e \in \Sigma$ be the word resulting form γ, if we replace t_1 by a_1, t_2 by a_2, … up to t_e by a_e. We define the **emission probabilities**

$$P(a_1,\dots,a_e \mid A \to \gamma) := \begin{cases} \frac{P(A\to\gamma^{(j)})}{P(A\to\gamma)} & \text{if } \gamma(a_1,\dots,a_e) = \gamma^{(j)} \\ 0 & \text{otherwise} \end{cases}.$$

So, the emission probability $P(a_1,\dots,a_e \mid A \to \gamma)$ is the probability that — given an occurrence of rule tempate $A \to \gamma$ — the instance of the template corresponding to the terminals $a_1,\dots,a_e$ is selected; which is 0 if no such instance exists. Conceptually, this divides the process of rule selection into two subsequent parts: first a rule template is selected and then an instance of this template.

This division does not change the stochastic model compared to not using rule templates: $P(a_1,\dots,a_e \mid A \to \gamma)$ is the *conditional* probability for terminals $a_1,\dots,a_e$ *given* rule template $A \to \gamma$. This means, we may have *different* emission probabilities for every rule template and the product of transition and emission probability is again the original probability of the instance rules: $P(a_1,\dots,a_e \mid A \to \gamma)\cdot P(A \to \gamma) = P(A \to \gamma^{(j)})$ iff $\gamma(a_1,\dots,a_e) = \gamma^{(j)}$.

In the implementation of our parser, we will not use the splitting into transition and emission probabilities, therefore we will write

$$\begin{aligned} P(A \to \gamma \wedge a_1,\dots,a_e) &:= P(A \to \gamma)\cdot P(a_1,\dots,a_e \mid A \to \gamma) \\ &= \begin{cases} P(A \to \gamma^{(j)}) & \text{if } \gamma(a_1,\dots,a_e) = \gamma^{(j)} \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

for the **joint probability** of rule template $A \to \gamma$ and terminals $a_1,\dots,a_e$. Note that this probability is defined for all choices of terminals, even if according rules do not exist — in which case it is 0.

If it is convenient for the discussion, we will regard a single rule as a rule template on its own, comprised of just one instance. In rule templates of order 0 — i. e. rules without terminal symbols — rule template and rule instance *coincide*, always yielding an emission probability of one.

3.1.5. Rule Templates in the Context of Secondary Structure Prediction

Grammars for secondary structure prediction use derivations of primary structure to encode secondary structure. Although in general no restrictions on this encoding are imposed other than non-ambiguity, most approaches use a simple encoding that nicely fits the concept of rule templates:

Instead of the generic terminal placeholder t from the last section, one uses the symbols from the bar-bracket-representation of (joint) secondary structure — with the same meaning: `|` represents unpaired bases, matching `( )` and `[ ]` stand for internal and external bonds, respectively. For a given leftmost derivation of a primary structure, one can easily determine the symbol in the primary structure each placeholder was substituted with. The (joint) secondary structure encoded by that derivation is simply the set of index pairs matching parentheses correspond to. This way, we do not need additional information about the encoding of secondary structures.

The notion of 'transition' and 'emission' probabilities introduced in the last section did not change the stochastic model, as we had *distinct* emission probabilities for every rule template.

For our implementation of probability training, we will *drop* this dependency on the rule template. Rather, we will use *one* probability for every group of terminal placeholders and possible combination of terminals for those placeholders, i. e. we will have four probabilities $\Pr[\texttt{|} \to x]$ for $x \in \{a, c, g, u\}$ and sixteen probabilities $\Pr[\texttt{( )} \to xy]$ and $\Pr[\texttt{[ ]} \to xy]$ for $x, y \in \{a, c, g, u\}$. Of course, this does change our stochastic model, but not every change is bad:

- Especially for RNA-RNA joint secondary structures, quite few structures are known that can be used to train our model; a reduction in the degrees of freedom will help compensate for this lack of information.
- There is no convincing reason, why the different positions the placeholders occur in our grammars should behave 'very' different.[27] Differentiating between subtleties might even introduce over-fitting effects, thereby making predictions *worse*.

3.1.6. Inside and Outside Probabilities

Again, we could use inside and outside probabilities as defined for SMCFGs, but simplification is at hand — namely since non-terminal components always stay within 'their' dimensions. For $w \in \Sigma^{n_1} \times \cdots \times \Sigma^{n_m} \subset (\Sigma^\star)^m$, $A \in N$ and indices

$$\begin{gathered} 1 \leqslant l_1 \leqslant r_1 \leqslant n_1 + 1 \\ \vdots \\ 1 \leqslant l_m \leqslant r_m \leqslant n_m + 1 \end{gathered}$$

we define

$$\alpha(A\,;\, l_1, r_1, \ldots, l_m, r_m) \quad := \quad \Pr\left[A \Rightarrow^* \begin{pmatrix} (w_1)_{l_1, r_1 - 1} \\ \vdots \\ (w_m)_{l_m, r_m - 1} \end{pmatrix}\right] \qquad \text{and}$$

$$\beta(A\,;\, l_1, r_1, \ldots, l_m, r_m) \quad := \quad \Pr\left[S \Rightarrow^* \begin{pmatrix} (w_1)_{1, l_1 - 1} \\ \vdots \\ (w_m)_{1, l_m - 1} \end{pmatrix} \cdot A \cdot \begin{pmatrix} (w_1)_{r_1, n_1} \\ \vdots \\ (w_m)_{r_m, n_m} \end{pmatrix}\right] .$$

[27] Actually there is some knowledge, that could make use of rule-dependent emission probabilities. For example, a cg-pair is slightly more stable than other pairs, since it can form three hydrogen bonds, whereas au- and gu-pairs only have two bonds. This increased stability seems to have a greater effect at the start or end of a stem than in the middle of it. So, it might be expected to find cg with slightly bigger probability at the rims of a stem. However, our currently used grammars could only treat the *start* of a stem differently from the rest; they are not able to tell whether a created pair is the end of a stem or not. So if we made the emission probabilities rule-dependent, our model became asymmetric and would need quite some more parameters to be trained.

3.2. A Grammar for RIP

3.2.1. Handling two molecules

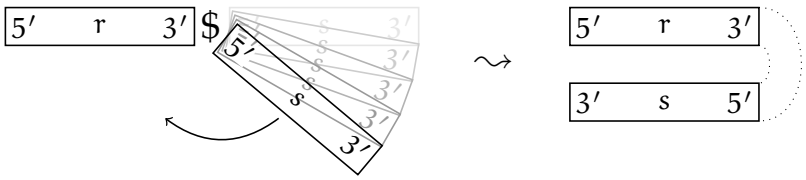
The classical approach to RIP uses a *single* word ‘r$s’, with primary structure r followed by a separator $ and the primary structure s, both in 5′–3′ direction.

This concatenation approach allows for a bijection from joint structures without internal and external pseudoknots to a subclass of correctly parenthesized dot-bracket-words with two types of brackets, ‘()’ and ‘[]’[28] and a dollar symbol somewhere in between. We consider round parentheses to correspond to internal pairs and square brackets to represent external base pairs. The subclass adds the following two constraints:

- ‘[’ is only allowed to the left of $ and ‘]’ only to the right.
- Two corresponding ‘(’ and ‘)’ may not enclose $ and have distance at least 4.

The grammar we will use produces *2-tuples*, i. e. *pairs* of words — it will be a *2-dimensional* context-free grammar. Since RIP actually addresses *two* distinct sequences, this approach is in a way more natural than separating the two sequences by the artificial marker $.

Our order of arrangement of the two primary structures r and s might seem counter-intuitive at first: We produce the first molecule r in typical 5′–3′ order; the second molecule s is generated in 3′–5′ direction, i. e. mirrored. However, this ‘reversal’ of s is what actually happens in nature[29] and is equivalent to above mentioned concatenation approach — if viewed from the right perspective: Flipping r downwards around $ gives exactly the tuple generated by our grammar.



Therefore, we can stick to the bar-bracket representation as given above, but the output of our grammar needs to be flipped back: The first element of the pair gets concatenated with the *reverse* of the second one. Of course, this leads to internal pairs in r being created as ‘)(’ (see the rules for J in Grammar 1), which is admittedly awkward to look at … sorry for that, but we will stick to it, nevertheless. That way, we can simply output the reverse of the second part without rearranging parentheses.

[28] More formally: The shuffle-product of correctly parenthesized dot-bracket words, where one is only using round parentheses and the other only square brackets.

[29] All known RNA-RNA interactions contain rather long regions of *stacked* external pairs and known results for secondary structures of single RNA molecules also suggest the outstanding stabilizing effect of stems. Each side of the stem winds into a helix and helices of facing stem sides *fit into* each other, like a natural screw thread. However, only helices ‘starting’ at opposite sides in the primary structure seem to fit. Looking at plain secondary structures *all* non-knotted stems are actually of this kind; hairpins in between being needed changes of direction. So, after some mental twists, our intuition turn out screwed up: reversing s is not quirky, but natural!

3.2.2. G_{RIP}

The idea for the grammar we use is due to [HQRS10, HQRS09], wherein the authors propose a partitioning scheme, that allows unique decomposition of zig-zag- and pseudoknot-free joint structures. The scheme is best explained graphically, so see figure 3.1 on the facing page.

While the recursive decomposition is essentially comprised of context-free rules and hence directly transferable into a 2D-CFG, there is one property of the decomposition inexpressible in the world of context-free grammars: Hybrids and secondary structure segments are required to be *maximal*, i. e. not extensible in the *context* of their usage ... obviously contradicting context-*free*ness if not further precautions are taken.

However, simply ignoring this requirement is not viable: Since the secondary structure segments may be empty, the following leftmost derivations are legal (using in advance the notation for 2D-CFG and the letters from the bottom of figure 3.1)

$$\mathsf{E} \Rightarrow \mathsf{K} \Rightarrow \binom{\mathsf{L}_1\mathsf{M}[}{\mathsf{L}_2\mathsf{M}]} \Rightarrow \binom{\mathsf{L}_1[}{\mathsf{L}_2]} \Rightarrow \binom{[\,[}{]\,]}$$
$$\mathsf{E} \Rightarrow \mathsf{KD} \Rightarrow \binom{[\,\mathsf{D}_1}{]\,\mathsf{D}_2} \Rightarrow \binom{[\,\mathsf{A}\mathsf{E}_1}{]\,\mathsf{B}\mathsf{E}_2} \Rightarrow \binom{[\,\mathsf{E}_1}{]\,\mathsf{E}_2} \Rightarrow \binom{[\,\mathsf{K}_1}{]\,\mathsf{K}_2} \Rightarrow \binom{[\,[}{]\,]}\,.$$

(Note that the second derivation violates the maximality of K.)
As this little example shows, the grammar resulting from naïve translation is *ambiguous* on secondary structures and therefore *not* suited for our application:

We compute the most likely derivation and would like to conclude that the structure represented by this derivation is consequently the most likely (joint) secondary structure. This does not hold, if there may be several different derivations for a single secondary structure, since the sum of several derivations might outweigh a single most likely derivation. Apart from the cold shiver a theorist feels running down his spine faced with a formally inapplicable heuristic, Dowell and Eddy observe in [DE04] that such ambiguity in prediction grammars does cause problems in practical applications, as well, and should not be ignored.

To avoid ambiguity in G_{RIP}, we deviate from the decomposition shown in figure 3.1 by defining for this thesis:

> Hybrids are *maximal* regions of stacked external pairs, **without any** unpaired bases.

... but wait; didn't all the trouble start with exactly this kind of maximality constraint — which is still used in our definition? Indeed, disallowing interruptions of hybrids by totally unpaired regions is not quite enough. But if every K is surrounded by structures, that are neither empty nor another hybrid, then K will for sure represent a maximal region, as required by our new definition of hybrids.

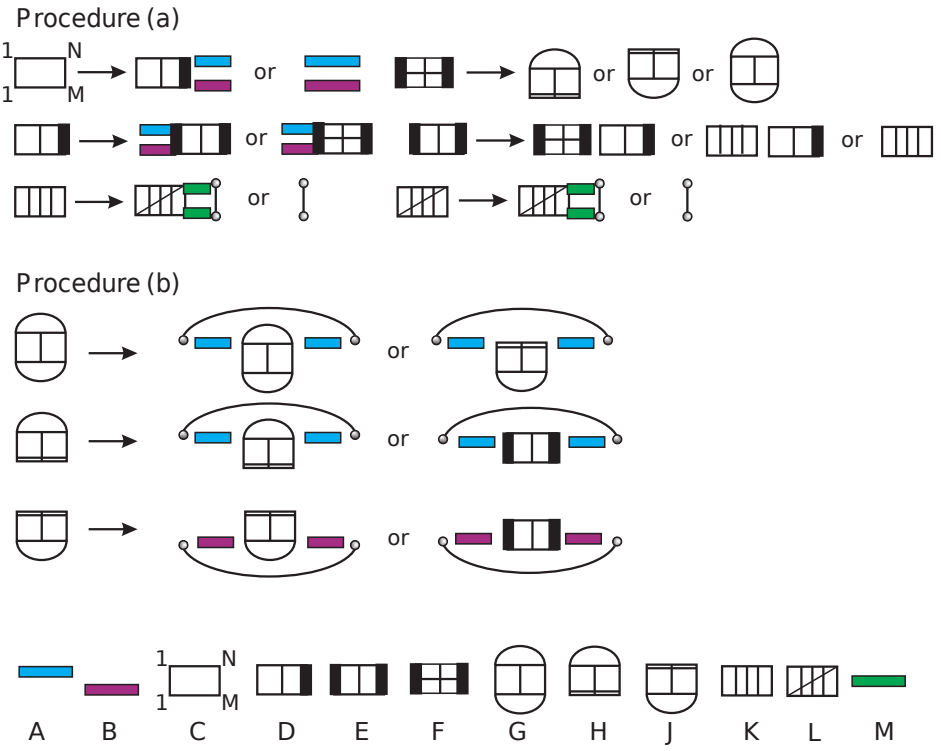


Figure 3.1.: The partitioning scheme 'rip2' proposed by HUANG et al. in [HQRS10, figure 4]. The start symbol is found in the left upper corner. Blue, pink and green† bars (**A**, **B** and **M**) represent regions of arbitrary length *without external* bonds. Blue and pink may contain *internal* pairs in r and s, respectively; green regions are totally unpaired. All three may stand for *empty* regions, as well. HUANG et al. introduce the following names for substructures:

D: right tight, **E**: double tight, **F**, **G**, **H**, **J**: tight, **K**: hybrid.

An important note given by the authors is that the secondary structures and hybrids are *maximal*, i. e. cannot be elongated to the left or right.

The graphically represented partitioning steps bear an apparent resemblance to context-free grammar rules. Since this scheme operates on *pairs* of primary structures and creates internal and external bonds in inter-meshed fashions, it cannot be represented as plain CFG — however, 2D-CFGs are ideally suited for that purpose (see Grammar 1 on page 39).

† For those of us who only got a lousy black and white copy of figure 3.1: The green bars appear only in rules for hybrids, i. e. in the third row. In all other rules, bars in the upper dimension are blue, those in lower dimension are pink.

Assuming that is guaranteed, above definition is the only deviation from substructures of [HQRS10]. But we still have to ensure in our grammar, that KK is never derivable.[30] To achieve this, we use the following non-terminals. For ease of understanding, we give explicative 'semantics' for them:

	in r	in s	in $\binom{r}{s}$
non-empty secondary structure	A	B	C
possibly empty secondary structure	X_A	X_B	X

We use X in all places rip2 had $\binom{A}{B}$ — except for rule $E \to KCE$: Since $E \Rightarrow K$, we need a non-empty 'separator' structure to avoid derivation of KK. As that changed rule also produced structures other than $K\binom{A}{B}K$, some more rules for E are needed to re-allow these combinations without secondary structure in between. Furthermore, we reverse the right hand sides of the rules for K and L to avoid left-recursion.[31] Apart from that, translation from rip2 to G_{RIP} is straight-forward and we finally arrive at Grammar 1 on the next page, a secondary-structure-unambiguous 2D-CFG for joint secondary structure prediction.

The unambiguity of G_{RIP} for joint secondary structures now follows from the proofs given by Huang et al. and our discussion above.

3.3. Secondary Structure Subgrammars

In the last section we proposed G_{RIP}, a 2D-CFG for two interacting RNA molecules. G_{RIP} contains two non-terminals with effective dimension one, namely A and B. These denote secondary structure regions in r and s, respectively, without external pairs.

We give the expansions for A and B as one-dimensional subgrammars. Let the one for A be called G^{SecStr} and the one for B G_{SecStr}. G^{SecStr} and G_{SecStr} will be almost identical, such that we can train G^{SecStr} in isolation and use the computed probabilities for G_{SecStr}, as well. This makes sense as long as the considered RNAs r and s are of similar type, which is assumed here for simplicity. If a specific class of interacting RNAs requires different training for r and s, we additionally train G_{SecStr} in isolation and use those probabilities.

3.3.1. G^{SecStr} — secondary structures in r

Research in the field of secondary structure prediction via SCFGs has lead to a whole zoo of grammars usable for structure prediction. For instance, some rather small examples can be found in [DE04]. However, to the knowledge of the author there are no studies available that compare efficiency of grammars in the context of RIP, namely predicting the parts without external bonds occurring inside an interaction complex. G^{SecStr} therefore was chosen according to the following reasoning:

[30] The second derivation from the example above would have had sentential form KK if we postponed the expansion of the first K to the very end.

[31] For these simply rules it is rather easy to see that this does not change the stochastic model implied by G_{RIP}. Lemma 3, discussed in detail in section 3.3.2, provides a formal argument.

Grammar 1 The 2D-CFG G_{RIP} used for modelling RNA-RNA joint secondary structures.

$$G_{RIP} = \Big(\{S, D, E, F, K, L, G, H, J, C, X, X_A, X_B, A, B\},\ \{a, c, g, u\},\ R,\ S,\ P\Big)$$

$$\begin{array}{lll}
S \to C & & S \to DX \\
D \to XE & & D \to XF \\
E \to FD & E \to KFD & E \to KCE \\
 & E \to KXF & E \to K \\
F \to G & F \to J & F \to H
\end{array}$$

$$\begin{array}{ll}
K \to \left(\begin{smallmatrix}[\\]\end{smallmatrix}\right) & K \to \left(\begin{smallmatrix}[& L_1\\] & L_2\end{smallmatrix}\right) \\
L \to \left(\begin{smallmatrix}[\\]\end{smallmatrix}\right) & L \to \left(\begin{smallmatrix}[& L_1\\] & L_2\end{smallmatrix}\right) \\
G \to \left(\begin{smallmatrix}(& X_A & G_1 & X_A &)\\ & & G_2 & & \end{smallmatrix}\right) & G \to \left(\begin{smallmatrix}(& X_A & J_1 & X_A &)\\ & & J_2 & & \end{smallmatrix}\right) \\
H \to \left(\begin{smallmatrix}(& X_A & H_1 & X_A &)\\ & & H_2 & & \end{smallmatrix}\right) & H \to \left(\begin{smallmatrix}(& X_A & E_1 & X_A &)\\ & & E_2 & & \end{smallmatrix}\right) \\
J \to \left(\begin{smallmatrix} & & J_1 & & \\ ) & X_B & J_2 & X_B & (\end{smallmatrix}\right) & J \to \left(\begin{smallmatrix} & & E_1 & & \\ ) & X_B & E_2 & X_B & (\end{smallmatrix}\right)
\end{array}$$

$$\begin{array}{ll}
X \to \left(\begin{smallmatrix}\varepsilon\\ \varepsilon\end{smallmatrix}\right) & X \to C \\
X_A \to \left(\begin{smallmatrix}\varepsilon\\ \varepsilon\end{smallmatrix}\right) & X_A \to \left(\begin{smallmatrix}A\\ \varepsilon\end{smallmatrix}\right) \\
X_B \to \left(\begin{smallmatrix}\varepsilon\\ \varepsilon\end{smallmatrix}\right) & X_B \to \left(\begin{smallmatrix}\varepsilon\\ B\end{smallmatrix}\right) \\
C \to \left(\begin{smallmatrix}A\\ \varepsilon\end{smallmatrix}\right) \quad C \to \left(\begin{smallmatrix}\varepsilon\\ B\end{smallmatrix}\right) & C \to \left(\begin{smallmatrix}A\\ B\end{smallmatrix}\right)
\end{array}$$

$+$ subgrammars for A and B,
namely G^{SecStr} and G_{SecStr} (see section 3.3)

We use the 2-dimensional context-free grammar G_{RIP} that creates pairs of words, resembling the 'stacking' $\left(\begin{smallmatrix}r\\ s\end{smallmatrix}\right)$, i. e. we will generate r in the first component of a 2-tuple and s in the second one. The first molecule, r, is produced in (typical) $5'-3'$ order, the second molecule, s, is generated in $3'-5'$ order (reversed).
All non-terminals except for A, B, X_A and X_B have effective dimension *two*, therefore operating on the joint structure, $D_{\neq\varepsilon}(A) = d_{\neq\varepsilon}(X_A) = \{1\}$ and $D_{\neq\varepsilon}(B) = d_{\neq\varepsilon}(X_B) = \{2\}$. The rules are given in a mixed form of vector and expanded form, whichever is more convenient. Every occurrence of a non-terminal C may be replaced by the expanded vector $\left(\begin{smallmatrix}C_1\\ C_2\end{smallmatrix}\right)$. Remember as well (or see section 3.1.2), that the expression $\left(\begin{smallmatrix}A\\ B\end{smallmatrix}\right)$ is just a shorthand for AB — or equivalently $\left(\begin{smallmatrix}A_1 & B_1\\ A_2 & B_2\end{smallmatrix}\right)$ — since we know that A_2 and B_1 can derive nothing else than ε.
The alphabet of G_{RIP} contains a, c, g and u, although none of these appear in the rules. Instead, the bracket symbols '(', ')' and '[', ']' are used as *placeholders* for these terminals: A matching pair of these brackets stands for a base pair, that forms an internal or external bond, respectively. Unpaired bases are only produced via G^{SecStr} or G_{SecStr}.

- A typical trade-off concerning the size of the grammar — measured in number of non-terminals, rules or overall length of right hand sides — states: Larger grammars are slower and need more training data, but provide more details hence have potential for better predictions.

  We use G^{SecStr} as preprocessing for parsing in G_{RIP} (see section 3.5 on page 48) on the same input — so, overall runtime and memory consumption are dominated by the two-dimensional parts. Additionally, plenty training data for plain RNA secondary structure is available. Consequently, if we have the slightest reason to believe of a larger grammar to improve upon prediction results, we should use the large grammar.

- Having further applications in mind — namely predicting by statistical sampling instead of most likely derivations (see section 6.1.1) — we choose a grammar suitable for these future applications.

For those reasons, we chose the rather huge grammar shown in Grammar 2 proposed by Nebel and Scheid in [NS10]. This grammar has proven to appropriately model RNA secondary structures, provided enough training data. It follows the decomposition realized by McCaskill for the partition function in [McC90].

3.3.2. G_{SecStr} — secondary structures in s

Now that we have a suitable grammar for secondary structures, we need a grammar for the *second* RNA molecule s — remember that s is produced in *reverse*! So simply setting $G_{SecStr} := G^{SecStr}$ does not work.

Our parser will deal with the one-dimensional subgrammars as preprocessing, so we could simply do this preprocessing for the reversed s. However this requires index transformations in the 2D-parser that would be specific to this application. Fortunately, there is a much more elegant solution: $G_{SecStr} := (G^{SecStr})^R$, meaning all right hand sides of rules in G^{SecStr} are reversed.

This is OK from the theoretical point of view, but introduces left-recursion in G_{SecStr}, which our implementation of the parser *cannot* handle — motivations of this limitation are discussed in chapter 4. However G^{SecStr} has a very handy property, allowing us to remove the left-recursion without 'affecting' the implied stochastic model. Before we do that, let us formalize this needed equivalence of G_{SecStr} and $(G^{SecStr})^R$.

Definition: Two SCFGs G and G' are called **stochastically equivalent**, if a bijection b from complete[32] parse trees in G to complete parse trees in G' exists, such that for all complete parse trees T in G holds:

(1) The frontier of T is equal to the frontier of $b(T)$.

(2) $\Pr[T] = \Pr[b(T)]$

Stochastic equivalence obviously implies ordinary equivalence. Using this notation, we are looking for modifications of $(G^{SecStr})^R$ to remove left-recursion while retaining stochastic equivalence. The following lemma provides the basis for that.

[32] Complete parse trees are parse trees whose frontier contains only terminals.

Grammar 2 The SCFG G^{SecStr} used for modelling single RNA secondary structures.

$$G^{\text{SecStr}} = \Big(\{S, T, C, A, P, L, F, H, G, B, M, O, N, U, Z\},\ \{a, c, g, u\},\ R,\ S,\ P\Big)$$

$$\begin{aligned}
S &\to T \\
T &\to C \qquad T \to A \qquad T \to CA \qquad T \to AT \qquad T \to CAT \\
C &\to ZC \qquad C \to Z \\
A &\to (^{m}\, L\,)^{m} \\
P &\to (\, L\,) \\
L &\to F \qquad L \to P \qquad L \to G \qquad L \to M \\
F &\to Z^{k-1}\, H \\
H &\to ZH \qquad H \to Z \\
G &\to BA \qquad G \to AB \qquad G \to BAB \\
B &\to ZB \qquad B \to Z \\
M &\to UAO \\
O &\to UAN \\
N &\to UAN \qquad N \to U \\
U &\to ZU \qquad U \to \varepsilon \\
Z &\to |
\end{aligned}$$

This grammar was proposed by NEBEL and SCHEID in [NS10, definition 2.3], and was originally designed for statistical sampling. Two parameters $m \in \mathbb{N}_{\geqslant 1}$ and $k \in \mathbb{N}_{\geqslant 1}$ are given to adapt the grammar:

m : minimal stem length $\qquad$ k : minimal hairpin size

Typical values are $m = 1$ and $k = 3$. Note that m and k are *constants* in any application of this grammar.

Lemma 3: Given a SCFG $G = (N, \Sigma, R, S, P)$ let us define SCFG $G_A := (\{A\}, N_A, R_A, A, P)$ for a non-terminal $A \in N$, where R_A is the set of all rules with left hand side A and $N_A \subset N \cup \Sigma$ is the set of all symbols occurring in those right hand sides, but without A itself. Now let $G'_A = (\{A\}, N_A, R'_A, A, P'_A)$ be an arbitrary grammar equivalent to G_A, i. e. $\mathcal{L}(G_A) = \mathcal{L}(G'_A)$. Then, grammar G', defined by

$$G' := (N, \Sigma, (R \backslash R_A) \dot\cup R'_A, S, P') \qquad \text{with} \qquad P'(r) := \begin{cases} P(r) & \text{if } r \in R \backslash R_A \\ P'_A(r) & \text{if } r \in R'_A \end{cases},$$

is equivalent to G, i. e. fulfills $\mathcal{L}(G) = \mathcal{L}(G')$. Moreover, if G_A and G'_A are unambiguous and stochastically equivalent, then G and G' are stochastically equivalent, as well.

Proof: Given a complete derivation tree T for $S \overset{*}{\underset{G}{\Rightarrow}} w$. If no A appears in T, this derivation is identically usable in G' and we are done.

So, assume there is an A in the tree and let T_A be the subtree rooted by a topmost occurrence of A, i. e. on the path from that A to root S, no other As are located. Now remove in T_A the child trees of all nodes *not* labeled A. The remaining tree $\hat{T}_A$ is a partial derivation $A \overset{+}{\underset{G}{\Rightarrow}} \alpha$ for some $\alpha \in ((N \cup \Sigma) \backslash \{A\})^\star$ that uses only rules with left hand side A. Therefore, we have $\alpha \in \mathcal{L}(G_A) = \mathcal{L}(G'_A)$, which implies the existence of a partial derivation tree $\hat{T}'_A$ for $A \Rightarrow^+ \alpha$ in G'_A. Replacing $\hat{T}_A$ by $\hat{T}'_A$ in T and iterating this procedure until all occurrences of A are expanded using rules from G'_A yields a parse tree T' for $S \Rightarrow^* w$ in G', so we have $\mathcal{L}(G) \subseteq \mathcal{L}(G')$. Since the above procedure works symmetrically, we get $\mathcal{L}(G) \supseteq \mathcal{L}(G')$ by the same arguments, hence $\mathcal{L}(G) = \mathcal{L}(G')$.

For the second claim, let b_A be the bijection from complete parse trees in G_A to the ones in G'_A given by stochastic equivalence. Again, we use the same procedure as above, but notice:

(1) $\hat{T}_A$ is the one and only derivation for $A \Rightarrow \alpha$ in G_A, since G_A is unambiguous.

(2) Setting $\hat{T}'_A := b_A(\hat{T}_A)$ uniquely determines the replacement tree $\hat{T}'_A$ and both trees have the same probability.

Therefore, we get for each complete parse tree T in G a uniquely determined parse tree T' in G' with identical frontier and probability. So, our procedure serves as bijection between parse trees in G and G', proving their stochastic equivalence. □

For secondary structure prediction, stochastic equivalence is only a *necessary* condition: We also need parse trees declared equivalent by b to encode the *same* (partial) secondary structure. As our encoding only depends on rules containing `(` and `)`, for most choices for A from the lemma, stochastic equivalence is actually sufficient — namely, when no parentheses show up in $\mathcal{L}(G_A)$. This leaves us with A and P, which will be dealt with separately.

To apply our lemma to $(G^{\text{SecStr}})^R$, we start by determining the languages $\mathcal{L}(G_A)$ for G^{SecStr}, see table 3.1. Then, we need to find grammars for those languages that are unambiguous and stochastically equivalent to the rules in $(G^{\text{SecStr}})^R$.

A	$\mathcal{L}(G_A)$	A	$\mathcal{L}(G_A)$	A	$\mathcal{L}(G_A)$
S	T	L	$F + P + G + M$	M	UAO
T	$C + \left(C^? A\right)^+ C^?$	F	$Z^k Z^*$	O	UAN
C	Z^+	H	Z^+	N	$\left(UA\right)^+ U$
A	$(^m\, L\,)^m$	G	$BA + AB + BAB$	U	Z^*
P	$(\, L\,)$	B	Z^+	Z	$\vert$

Table 3.1.: The languages $\mathcal{L}(G_A)$ from Lemma 3 for grammar G^{SecStr}, given as extended regular expressions. Note explicitly, that the language for A consists of only *one* word, as m is a constant. Large parentheses $\left(\,\right)$ are used for grouping regular expressions, whereas the small ones $(\ )$ are literals. x^+ means at least one copy of x and $x^?$ stands for one or zero occurrences of x. Notice that non-terminals are used as literals in these regular expression.

Noticeably, all those languages are *regular*.[33] Additionally, we observe that most languages are *palindromes*, i. e. they are indifferent to reversal. The only exceptions are:

- $(^m\, L\,)^m$ and $(\, L\,)$ resulting from A and P.
  These become $)^m\, L\, (^m$ and $)\, L\, ($ on reversal. However, the parentheses are only terminal placeholders, indicating internal bonds (see section 3.1.5)! So if we *exchange* the emission probabilities of corresponding base pairs — emission probability of au need not be equal to the one of ua — we are done.

  Reversing $(^m\, L\,)^m$ and $(\, L\,)$ does *not crucially* affect encoded secondary structure: As we encode secondary structure as a *set of pairs*, reversing the order *inside* one such pair does only formally change the structure. However, we produce s in reverse order anyway, so it actually makes *sense* to produce the secondary structure in reverse, as well.

- UAO and UAN resulting from M and O.
  Considering the four rules creating a multiloop $M \to UAO$, $O \to UAN$, $N \to UAN$ and $N \to U$ in a single grammar

$$G_{MNO} := \Big(\{M, N, O\}, \{U, A\}, \{M \to UAO, O \to UAN, N \to UAN, N \to U\}, M\Big)\,,$$

  we get $\mathcal{L}(G_{MNO}) = UA\,(UA)^+\,U$, which is a palindrome language.

Every form of repetition involved in the languages of table 3.1 is created in G^{SecStr} *unambiguously* by plain *right-recursion* — comprised of two types of rules: (a) right-recursive rules, i. e. rules of form $A \to \alpha A$ with $A \notin \alpha \wedge \alpha \neq \varepsilon$ and (b) rules terminating this repetition, i. e. $A \to \beta$ with $A \notin \beta$.

As $|R_M| = |R_O| = 1$, the probability for $M \to UAO$ and $O \to UAN$ must be 1, leaving us with one rule for extension and one rule for stopping in the case of G_{MNO}, as well.

[33] Of course, this is not the case for arbitrary CFGs: In $G_1 = \big(\{S\}, \{a, b\}, \{S \to aSb, S \to \varepsilon\}, S\big)$, $\mathcal{L}(G_S) = \mathcal{L}(G_1) = \{a^n b^n : n \geqslant 0\}$, which is known to be *not* regular. However, every grammar can easily be *converted* into such a grammar, by replacing all uses of non-terminal A in rules expanding A by a new non-terminal A' with a single rule $A' \to A$. The new grammar is obviously equivalent to the old one. This provides a nice proof that removability of left-recursion in regular grammars carries over to context-free grammars.

The probability of a word created that way only depends on the *number* each type of repetitive rule is used and the *type of stop* case, because of commutativity of multiplication.

In $(G^{\text{SecStr}})^R$, the right hand sides of rules are reversed, thereby giving unambiguous *left*-recursive generation. But since the languages are palindromes and the probability only depends on the number of repetitions, not the exact order of creation, we can indeed use *exactly* the rules of G^{SecStr} for G_{SecStr} — avoiding left-recursion by reversing the right hand sides of rules in $(G^{\text{SecStr}})^R$ once again.

the devil is in the details ...

Simply setting $G_{\text{SecStr}} := G^{\text{SecStr}}$ still does not work, two things need to be done when creating G_{SecStr} from G^{SecStr}:

(1) The emission probabilities for pairs have to be swapped: If xy for $x, y \in \Sigma$ is created with probability p in G^{SecStr}, yx is created with probability p in G_{SecStr}.

(2) The probabilities for rules $G \to BA$ and $G \to AB$ — representing left and right bulges respectively — have to be swapped.

Yet, this is easily done and still avoids separate training of G^{SecStr} and G_{SecStr}.

3.4. Earley-Parser for 2D-CFGs

We adopt the item and inference rule based approach of describing parsers from [Goo98, Goo99] to present our Earley-style parser for 2-dimensional context-free grammars (see Algorithm 3.1 on the next page). We will in the following assume that

(1) we are parsing the terminal tuple $\binom{u}{v}$ with $u \in \Sigma^n$ and $v \in \Sigma^m$

(2) in the stochastic 2D-CFG $G = (N, \Sigma, R, S', P)$.

(3) There is only one rule for S', namely $S' \to S$, for a non-terminal $S \in N$.
S' does not show up in any right hand side in R.

(4) G does not contain non-terminals of effective dimension zero.
(Such non-terminals can simply be deleted.)

(5) **Terminals** do **not** occur in the **middle** of a rule,
i. e. for every rule $A \to \gamma \in R$ holds $\gamma_1, \gamma_2 \in \Sigma^\star \mathcal{C}^\star \Sigma^\star$.

(6) G is **free of left-recursion**, i. e. $\nexists A \in N \;\; A \overset{+}{\underset{G}{\Rightarrow}} A\alpha$

The last requirement implies loop-freeness and therefore ensures that all semiring buckets are non-looping. G_{RIP} trivially fulfills all requirements if we add a new non-terminal S' and $S' \to S$.

Of course, in general, Earley parsing can cope with grammars containing left-recursion. In fact, one well known advantage of Earley's parser is its efficient handling of left-recursion. There are two reasons why we decided not to allow left-recursion, anyhow. First, left-recursion complicates the bucketing/ordering of items — see subsection 4.2.2 for details. Second, our optimistic prediction levels out the formerly superior performance of left-recursion for dense grammars.

Disallowing left-recursion is not a grave restriction — the grammars we intend to use do not have left-recursion and it is avoidable in general — e. g. in Greibach normal form.[34]

Our parser can handle ε-rules without doing inefficient completer-predictor-loops. This deserves explicit emphasis, since efficiently dealing with ε-rules is non-trivial for Earley parsing (see e. g. [AH02], [GJ08]).

Theorem 4: The invariant of our parser is

$$\binom{i \;\; j}{k \;\; l}, A \to \overset{\alpha_1 \bullet \beta_1}{\alpha_2 \bullet \beta_2} \text{ is derived} \qquad \text{iff} \qquad \binom{A_1}{A_2} \Rightarrow \binom{\alpha_1 \beta_1}{\alpha_2 \beta_2} \Rightarrow^* \binom{u_{i,j-1} \;\; \beta_1}{v_{k,l-1} \;\; \beta_2}. \tag{3.2}$$

Proof sketch: We did the correctness proof of classical Earley parsing in much detail (proof of theorem 1 on page 13 in section 2.2). We used the term of Earley-walks, a sequence of edges in parse trees mimicking the item derivation in the calculus of Earley's algorithm.

[34] Moreover, the transformation of a left-recursive SCFG G into a SCFG G' without left-recursion hinted at in footnote 33 on page 43 can easily be worked out to retain stochastic equivalence, if the grammar is loop-free, and can easily be generalized to mD-CFGs.

Algorithm 3.1 2D-EARLEY: an EARLEY-style parser for 2D-CFGs

► **Item Form:**

▷ normal items:
$\begin{pmatrix} i & j \\ k & l \end{pmatrix}, A \to \begin{smallmatrix} \alpha_1 \bullet \beta_1 \\ \alpha_2 \bullet \beta_2 \end{smallmatrix}\Big)$

▷ rule existence items:
Rule $\big[A \to \alpha\beta\big]$

satisfying
- ▷ $1 \leqslant i \leqslant j \leqslant n+1$
- ▷ $1 \leqslant k \leqslant l \leqslant m+1$
- ▷ α_1, α_2, β_1 and β_2 are possibly empty strings of terminals and non-terminals
- ▷ $A \to \left(\begin{smallmatrix} \alpha_1\beta_1 \\ \alpha_2\beta_2 \end{smallmatrix}\right)$ is a rule of the grammar

The items of shape Rule $\big[A \to \alpha\beta\big]$ are so-called **rule-existence items**. They exist right from the start and have as value the probability of their rule $P(A \to \alpha\beta)$.

► **Goal Item:** $\begin{pmatrix} 1 & n+1 \\ 1 & m+1 \end{pmatrix}, S' \to \begin{smallmatrix} S_1 \bullet \\ S_2 \bullet \end{smallmatrix}\Big)$

► **Inference Rules:**

▷ Start item:
$$\frac{}{\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}, S' \to \begin{smallmatrix} \bullet S_1 \\ \bullet S_2 \end{smallmatrix}\Big)}$$

▷ **Prediction** (optimistic):
$$\frac{\text{Rule}\Big[B \to \left(\begin{smallmatrix} \beta_1 \\ \beta_2 \end{smallmatrix}\right)\Big]}{\begin{pmatrix} j & j \\ l & l \end{pmatrix}, B \to \begin{smallmatrix} \bullet \beta_1 \\ \bullet \beta_2 \end{smallmatrix}\Big)}$$

▷ **Scanning:**

◇ in first component
$$\frac{\begin{pmatrix} i & j-1 \\ k & l \end{pmatrix}, A \to \begin{smallmatrix} \alpha_1 \bullet u_{j-1} \beta_1 \\ \alpha_2 \bullet \quad\quad \beta_2 \end{smallmatrix}\Big)}{\begin{pmatrix} i & j \\ k & l \end{pmatrix}, A \to \begin{smallmatrix} \alpha_1 u_{j-1} \bullet \beta_1 \\ \alpha_2 \quad\quad \bullet \beta_2 \end{smallmatrix}\Big)}$$

◇ in second component
$$\frac{\begin{pmatrix} i & j \\ k & l-1 \end{pmatrix}, A \to \begin{smallmatrix} \alpha_1 \bullet \quad\quad \beta_1 \\ \alpha_2 \bullet v_{l-1} \beta_2 \end{smallmatrix}\Big)}{\begin{pmatrix} i & j \\ k & l \end{pmatrix}, A \to \begin{smallmatrix} \alpha_1 \quad\quad \bullet \beta_1 \\ \alpha_2 v_{l-1} \bullet \beta_2 \end{smallmatrix}\Big)}$$

▷ **Completion:**
$$\frac{\begin{pmatrix} i & r \\ k & s \end{pmatrix}, A \to \begin{smallmatrix} \alpha_1 \bullet B_1 \gamma_1 \\ \alpha_2 \bullet B_2 \gamma_2 \end{smallmatrix}\Big) \quad \begin{pmatrix} r & j \\ s & l \end{pmatrix}, B \to \begin{smallmatrix} \beta_1 \bullet \\ \beta_2 \bullet \end{smallmatrix}\Big)}{\begin{pmatrix} i & j \\ k & l \end{pmatrix}, A \to \begin{smallmatrix} \alpha_1 B_1 \bullet \gamma_1 \\ \alpha_2 B_2 \bullet \gamma_2 \end{smallmatrix}\Big)}$$

The very same approach works here, as well; only two things change: First, all nodes now get two-dimensional labels: Either a non-terminal or a terminal expression $\binom{x}{y}$, where $x, y \in \{\varepsilon\} \cup \Sigma$ and $|xy| \leqslant 1$, i. e. at most one is a terminal.[35] This requires more cases to be distinguished, but otherwise causes no problems.

The second change stems from the simplified prediction rule: We do *not* require an item $\left(\begin{smallmatrix} i & j \\ k & l \end{smallmatrix}, C \to \begin{smallmatrix} \eta_1 \bullet B_1 \mu_1 \\ \eta_2 \bullet B_2 \mu_2 \end{smallmatrix}\right)$ as premise in the prediction rule, here: We use **optimistic prediction**. This means, we are always dealing with parse trees, where the root corresponds to the left hand side of the *current* rule. That makes the proof actually *easier*: The claim (3.2) is trivially fulfilled for $\alpha = \binom{\varepsilon}{\varepsilon}$ and for the other cases, the argumentation of the proof of theorem 1 remains valid. □

Application of theorem 4 to the goal item $\left(\begin{smallmatrix} 1 & n+1 \\ 1 & m+1 \end{smallmatrix}, S \to \begin{smallmatrix} S_1 \bullet \\ S_2 \bullet \end{smallmatrix}\right)$ proves the correctness of algorithm 3.1: The goal item is derived if and only if $\binom{u}{v} \in \mathcal{L}(G)$.

3.4.1. Item-related definitions

In order to work with the items, some terms and definitions turn out handy:

- First, we assign a unique **rule index** $\mathrm{ind}(r) \in \{1, \ldots, |R|\}$ to every rule $r \in R$. For the moment, we will only use the rule index to shortly write rules; later we use them to induce an order on the rules.
- We define the **dot-index** q of a dotted rule $A \to \binom{\alpha_1 \bullet \beta_1}{\alpha_2 \bullet \beta_2}$ by $q = \binom{q_1}{q_2} := \binom{|\alpha_1|}{|\alpha_2|}$. So q gives the position of the dot in a dotted rule by *counting* the number of characters *left* to the dot — ranging from zero to $|\alpha_1\beta_1|$ and $|\alpha_2\beta_2|$ respectively. Using these dot indices q and the grammar dependent rule index $p = \mathrm{ind}(A \to \alpha\beta)$, we can write an arbitrary item equivalently as

$$\left(\begin{smallmatrix} i & j \\ k & l \end{smallmatrix}, p, \begin{smallmatrix} q_1 \\ q_2 \end{smallmatrix}\right) = \left(\begin{smallmatrix} i & j \\ k & l \end{smallmatrix}, \mathrm{ind}(A \to \alpha\beta), \begin{smallmatrix} |\alpha_1| \\ |\alpha_2| \end{smallmatrix}\right) \quad \equiv \quad \left(\begin{smallmatrix} i & j \\ k & l \end{smallmatrix}, A \to \begin{smallmatrix} \alpha_1 \bullet \beta_1 \\ \alpha_2 \bullet \beta_2 \end{smallmatrix}\right).$$

- We call an operation producing an item I — an application of an inference rule, i. e. prediction, scanning or completion — **finalization**, if the conclusion item I has the dot at the very right end. Such an item I will be referred to as a **finalized** item.
- Given a sequence of items $I_1, \ldots, I_t$, such that

$$I_s = \left(\begin{smallmatrix} i & j_s \\ k & l_s \end{smallmatrix}, p, q^{(s)}\right) \qquad 1 \leqslant s \leqslant t,$$
$$\text{with } j_1 = i, \quad l_1 = k, \qquad q^{(1)} = \binom{0}{0}, \quad q^{(t)} = \text{length of right hand side},$$

  i. e. they only differ in the end and dot indices, starting at a predicted item I_1 and ending with a finalized item I_t. Assume further that I_{s+1} was derived from I_s by using *one* inference rule application ($1 \leqslant s < t$). Then we will sloppily talk about the whole sequence as **'one item'**, that is first **being predicted**, and then (repeatedly) **scanned** and/or **completed** and finally **finalized**.

[35] That is the portion of input our parser can handle in one scanning step.

3.5. One-dimensional preprocessing featuring an SCFG Earley parser

Subsection 3.1.3 on page 31 introduces the concept of *subgrammars*. In our case of 2D-CFGs the non-terminals not having 'full' dimension are *one*-dimensional. Therefore we will partition the non-terminals $N = N_2 \dot\cup N_1$ into the set N_2 of two-dimensional non-terminals and the set N_1 of one-dimensional ones.

The set $\mathcal{A}$ mentioned in subsection 3.1.3 hence contains only one-dimensional non-terminals, i. e. $\mathcal{A} \subseteq N_1$. So, the subgrammars G_A induced by $A \in \mathcal{A}$ are plain SCFGs!

This means, for the preprocessing of the subgrammars, we need a semiring parser for SCFGs. We will use an Earley-style parser *very* similar to the two-dimensional one; see Algorithm 3.2 on the next page. For this parser, we will assume that

(1) we are parsing the terminal word $w \in \Sigma^n$

(2) in the SCFG $G = (N, \Sigma, R, S', P)$.

(3) There is only one rule for S', namely $S' \to S$, $S \in N$.
S' does not show up in any right hand side in R.

(4) **Terminals** do **not** occur in the **middle** of a rule,
i. e. for every rule $A \to \gamma \in R$ holds $\gamma \in \Sigma^\star N^\star \Sigma^\star$.

(5) G is **free of left-recursion**, i. e. $\nexists A \in N \;\; A \overset{+}{\underset{G}{\Rightarrow}} A\alpha$

Theorem 5: For the rules from algorithm 3.2 on the facing page the following invariant holds:

$$\big(i \;\; j\,,\, A \to \alpha \bullet \beta\big) \text{ is derived} \qquad \text{iff} \qquad A \Rightarrow \alpha\beta \Rightarrow^* w_{i,j-1}\beta\,.$$

The proof is analogous to the one of theorem 4 on page 45, if you simply ignore the remark on two-dimensional node labels. □

Correctness of algorithm 3.2 again follows from applying theorem 5 to the goal item.

What remains, is the 'interface' between the two-dimensional parser 2D-Earley and this one — 1D-Earley. The only thing needed by 2D-Earley are *finalized* items — ones with the dot at the very right. Such items are needed in the completion rule to step over one-dimensional non-terminals. So we can simply add the following "interface rules" for the inclusion of non-terminal $A \in \mathcal{A}$:

Condition	Inference Rule
$D_{\neq\varepsilon}(A) = \{1\}$ and $w = u$	$\dfrac{\big(r \;\; j\,,\, A \to \gamma \bullet\big)}{\left(\begin{smallmatrix} r & j \\ l & l \end{smallmatrix}\,,\, A \to \begin{smallmatrix} \gamma\bullet \\ \bullet \end{smallmatrix}\right)}$
$D_{\neq\varepsilon}(A) = \{2\}$ and $w = v$	$\dfrac{\big(s \;\; l\,,\, A \to \gamma \bullet\big)}{\left(\begin{smallmatrix} j & j \\ s & l \end{smallmatrix}\,,\, A \to \begin{smallmatrix} \bullet \\ \gamma\bullet \end{smallmatrix}\right)}$

Algorithm 3.2 1D-Earley: an Earley-style parser for stochastic context-free grammars

- **Item Form:**
  - ▷ normal items: $(i\ \ j\,,\ A \to \alpha \bullet \beta)$
  - ▷ rule existence items: Rule $[A \to \alpha\beta]$

  satisfying
  - ▷ $1 \leqslant i \leqslant j \leqslant n+1$
  - ▷ α and β are possibly empty strings of terminals and non-terminals
  - ▷ $A \to \alpha\beta$ is a rule of the grammar

  The items of shape Rule $[A \to \alpha\beta]$ are so-called **rule-existence items**. They exist right from the start and have as value the probability of their rule $P(A \to \alpha\beta)$.

- **Goal Item:** $(1\ \ n+1\,,\ S' \to S \bullet)$

- **Inference Rules:**
  - ▷ Start item:
  $$\frac{}{(1\ \ 1\,,\ S' \to \bullet S)}$$
  - ▷ **Prediction** (optimistic):
  $$\frac{\text{Rule}\,[B \to \beta]}{(j\ \ j\,,\ B \to \bullet \beta)}$$
  - ▷ **Scanning:**
  $$\frac{(i\ \ j-1\,,\ A \to \alpha \bullet w_{j-1}\beta)}{(i\quad j\quad ,\ A \to \alpha w_{j-1} \bullet \beta)}$$
  - ▷ **Completion:**
  $$\frac{(i\ \ r\,,\ A \to \alpha \bullet B\gamma)\quad (r\ \ j\,,\ B \to \beta \bullet)}{(i\ \ j\,,\ A \to \alpha B \bullet \gamma)}$$

Remember, that we need probabilities for every substring of u and v, respectively, if we parse $\binom{u}{v}$ by 2D-Earley. But since we use optimistic prediction again, we predict items $\left(\begin{smallmatrix} i & i \end{smallmatrix}, S' \to \bullet S\right)$ for *every* start position i and will complete all of those for every end position j where such a completion is possible.

This means, we need only **one run** of 1D-Earley **per subgrammar**, and can extract all needed information from it — we do *not* have to parse *every substring* separately. The items corresponding to those substrings will be derived in the course of parsing the whole word, anyway. So, only with optimistic prediction is efficient preprocessing possible.

3.5.1. Outside probabilities in split grammars

Let us consider a simple case, where we have one 2D-CFG G_2 and a split-off 1D-CFG G_1, with start symbols S_2 and S_1, respectively. With inside probabilities, there is no discussion possible: We always have to fully expand everything to terminals — even if that means that a non-terminal from G_2 has to 'switch' into G_1 to complete its derivation. Otherwise, we cannot determine the indices in the terminal word produced.

However, splitting *has* consequences for the notion of outside probabilities: Shall the 'border' between the grammar be resembled in those values or do we like it to be transparent?

For outside probabilities of non-terminals in G_1, the question is: Start at S_2 or at S_1? Both choices are sensible — it depends on what we *use* those reverse values for. **We will let them start at S_1**, i. e. we define for a non-terminal A in G_1

$$\beta(A, i, j) := \Pr\left[S_1 \overset{*}{\Rightarrow} w_{1,i-1}\, A\, w_{j,|w|}\right] ,$$

because we introduced outside probabilities to **estimate rule probabilities** — and we will do this *separately* for the two-dimensional grammar and its one-dimensional subgrammars. Therefore we need the outside probabilities only *'inside'* of one grammar. Additionally, this allows us to use 1D-Earley as a 'standalone'-SCFG parser, e. g. for RNA secondary structure prediction.

3.6. Utilizing item values

So far, we have given an item-based parser description for parsing 2D-CFGs on one hand and on the other hand defined some things we are interested in — namely inside and outside probabilities, mainly for estimating rule probabilities — and Viterbi parses for prediction. We already mentioned that semiring parsing "does the trick" by assigning items of our parser *forward* and *reverse values* in a given semiring (see section 2.6). But we kind of weaseled out of unraveling how *exactly* those mysterious forward and reverse values are used — up to now.

The following fact is used in subsequent sections, so we state it here once and for all: In [Goo98, Goo99], Goodman chooses the reverse values in a way that for all items I the following holds

$$v(I) \otimes z(I) = \bigoplus_{\substack{D \text{ derivation}: \\ I \text{ appears in } D}} v(D) , \tag{3.3}$$

where the value of a derivation $\nu(D)$ is simply the $\otimes$-product of rule probabilities of all rule applications in the derivation D. The original definition takes the possibility into account, that one item may appear several times in one derivation. However, with non-looping buckets, this cannot happen, so we exclude this from our discussion.

3.6.1. Inside and outside probabilities

In this subsection, we will always assume the inside semiring $\left(\mathbb{R}_{\geqslant 0} \cup \{\infty\}, +, \cdot, 0, 1\right)$. The following equality is the core of the relation of inside probabilities and item values:

Theorem 6: For the (forward) values in the inside semiring holds for all $1 \leqslant i \leqslant j \leqslant n+1$ (and all $1 \leqslant k \leqslant l \leqslant m+1$ for the second claim)

$$\begin{aligned} \nu\left(i \;\; j\,,\, A \to \gamma \bullet\right) &= \Pr\left[A \Rightarrow \gamma \Rightarrow^* w_{i,j-1}\right] \quad \text{and} \\ \nu\left(\begin{smallmatrix} i & j \\ k & l \end{smallmatrix}\,,\, A \to \begin{smallmatrix} \gamma_1 \bullet \\ \gamma_2 \bullet \end{smallmatrix}\right) &= \Pr\left[A \Rightarrow \gamma \Rightarrow^* \binom{u_{i,j-1}}{v_{k,l-1}}\right], \end{aligned} \tag{3.4}$$

where $\nu(I) := 0$ if item I is not derivable at all.

Proof: We only prove the first claim; the second is shown analogously. The proof is done by induction on the *maximal height*[36] of a derivation tree for $A \Rightarrow \gamma \Rightarrow^* w_{i,j-1}$.

The basis is height 1, which implies $\gamma \in \Sigma^\star$. Then the right side is obviously $P(A \to \gamma)$ if $\gamma = w_{i,j-1}$, and 0 otherwise. If $A \not\Rightarrow^* w_{i,j-1}$, then $I := \left(i \;\; j\,,\, A \to \gamma \bullet\right)$ is not derivable by theorem 5. So, $\nu(I) = 0$ by definition. If $\gamma = w_{i,j-1}$, I is derivable — again by theorem 5. As γ only contains terminals, I can only be derived by scanning, which simply copies the premises item value to the conclusion, until we reach $\left(i \;\; i\,,\, A \to \bullet\gamma\right)$. This item is only derivable by prediction from Rule $\left[A \to \gamma\right]$ and therefore inherits its value, $P(A \to \gamma)$.

Now assume we have shown the assertion for all items such that the maximal height of a derivation tree for them is h and let $I := \left(i \;\; j\,,\, A \to \gamma \bullet\right)$ be an item whose maximal derivation height is $h+1$. Let $B_1, \ldots, B_k$ be the non-terminals occurring in γ. If one of the terminals does not match the corresponding input symbol, $A \not\Rightarrow^* w_{i,j-1}$, I is not derivable by theorem 5 and hence $\nu(I) = 0 = \Pr\left[A \Rightarrow \gamma \Rightarrow^* w_{i,j-1}\right]$. So, we only have to look at derivations, where the terminal parts fit. Let's call those derivations $D_1, \ldots, D_d$. According to the invariant of our parser, for every D_p and all $1 \leqslant q \leqslant k$, there is an item $I_{p,q}$ for the expansion of $B_q \Rightarrow \beta^{(p,q)} \Rightarrow^* w_{l_{p,q}, r_{p,q}}$ used in D_p. The sub-derivations for $I_{p,1}, \ldots, I_{p,k}$ have height $\leqslant h$, so the induction hypothesis applies:

$$\begin{aligned} \Pr\left[A \Rightarrow \gamma \Rightarrow^* w_{i,j-1}\right] &= P(A \to \gamma) \sum_{p=1}^{d} \prod_{q=1}^{k} \Pr\left[B_q \Rightarrow \beta^{(p,q)} \Rightarrow^* w_{l_{p,q}, r_{p,q}}\right] \\ &= P(A \to \gamma) \sum_{p=1}^{d} \prod_{q=1}^{k} \nu(I_{p,q})\,. \end{aligned}$$

But this is exactly the value of our item I: We started by predicting item $\left(i \;\; i\,,\, A \to \bullet\gamma\right)$ with value $P(A \to \gamma)$. By assumption, we have d ways to complete this item — assuming successful scanning. Each of those d ways contributes with the product of all items that were used for completion. □

[36] An item may have several distinct derivations. Therefore we use the maximum.

Now, we can express the inside and outside probabilities as defined in sections 2.5 and 3.1.6 in terms of forward and reverse values in the inside semiring. We first have a look at the simpler case of a SCFG and input word $w \in \Sigma^n$. Simply expanding the probabilities gives

$$\begin{aligned}\alpha(A,i,j) &= \Pr\left[A \Rightarrow^* w_{i,j-1}\right] \\ &= \sum_{A\to\gamma\in R_A} \Pr\left[A \Rightarrow \gamma \Rightarrow^* w_{i,j-1}\right] \\ &\underset{(3.4)}{=} \sum_{A\to\gamma\in R_A} \nu\big(i\ \ j\,,\, A\to\gamma\bullet\big)\,.\end{aligned}$$

When we multiply corresponding forward and reverse values we get

$$\begin{aligned}\nu\big(i\ \ j\,,\, A\to\gamma\bullet\big)\cdot z\big(i\ \ j\,,\, A\to\gamma\bullet\big) &\underset{(3.3)}{=} \sum_{\substack{D \text{ derivation}:\\ (i\ \ j\,,\,A\to\gamma\bullet) \text{ appears in } D}} \nu(D) \\ &= \Pr\left[S \Rightarrow^* w_{1,i-1}\,A\,w_{j,n} \Rightarrow w_{1,i-1}\,\gamma\,w_{j,n} \Rightarrow^* w\right] \\ &= \Pr\left[S \Rightarrow^* w_{1,i-1}\,A\,w_{j,n}\right]\cdot\Pr\left[A\Rightarrow\gamma\Rightarrow^* w_{i,j-1}\right] \\ &\underset{(3.4)}{=} \beta(A,i,j)\cdot\nu\big(i\ \ j\,,\,A\to\gamma\bullet\big)\,.\end{aligned}$$

This provides us with the following formula for outside probabilities:

$$\beta(A,i,j) = z\big(i\ \ j\,,\,A\to\gamma\bullet\big) \qquad \text{if } \nu\big(i\ \ j\,,\,A\to\gamma\bullet\big)\neq 0\,.$$

Since every application of forward and backward probabilities known to the author only considers the *product* of corresponding inside and outside probabilities, we content ourselves with this derivation.

Notice, that we do *not* have to sum over all rules expanding A for the outside probability, because the outer tree of an item $I := \big(i\ \ j\,,\,A\to\gamma\bullet\big)$ — as defined in [Goo98] — excludes everything derived from A at this position, so the outer value $z(I)$ *cannot* depend on the rule $A\to\gamma$ present in I.

For 2D-CFGs, all arguments directly carry over, so we simply state

$$\begin{aligned}\alpha(A;i,j,k,l) &= \sum_{A\to\gamma\in R_A} \nu\left(\begin{smallmatrix} i & j \\ k & l\end{smallmatrix}\,,\, A\to\begin{smallmatrix}\gamma_1\bullet\\ \gamma_2\bullet\end{smallmatrix}\right) \quad \text{and} \\ \beta(A;i,j,k,l) &= z\left(\begin{smallmatrix} i & j \\ k & l\end{smallmatrix}\,,\, A\to\begin{smallmatrix}\gamma_1\bullet\\ \gamma_2\bullet\end{smallmatrix}\right) \qquad \text{if } \nu\left(\begin{smallmatrix} i & j \\ k & l\end{smallmatrix}\,,\, A\to\begin{smallmatrix}\gamma_1\bullet\\ \gamma_2\bullet\end{smallmatrix}\right)\neq 0\,.\end{aligned}$$

3.6.2. Rule probability estimates

In section 2.5 on page 18, we mentioned the expectation maximization algorithm for training rule probabilities, when only primary structures are available as training data. In essence, one replaces the counted numbers of rule applications by *expected* numbers. As the latters depend on the current set of probabilities, an evolutionary algorithm results.

In [DEKM98, chapter 9], formulæ for these expected numbers are given in terms of inside and outside probabilities. However, it turns out that the kind of items we are dealing with offer a more direct way to compute those.

Let's again first look at a SCFG and input word $w \in \Sigma^n$. For a rule $A \to \gamma \in R$ we define $c(A \to \gamma)$ by

$$\begin{aligned} c(A \to \gamma) &:= \mathbb{E}\big[\#\text{rule applications } A \to \gamma\big] \\ &= \sum_{D \text{ derivation}} \nu(D) \cdot (\#\text{rule applications } A \to \gamma \text{ in } D) \\ &= \sum_{1 \leqslant i \leqslant j \leqslant n+1} \sum_{\substack{D \text{ derivation}: \\ (i\ j, A \to \gamma\bullet) \text{ appears in } D}} \nu(D) \\ &\underset{(3.3)}{=} \sum_{1 \leqslant i \leqslant j \leqslant n+1} \nu\big(i\ j, A \to \gamma\bullet\big) \cdot z\big(i\ j, A \to \gamma\bullet\big)\,. \end{aligned}$$

Analogously we define for 2D-CFGs and input tuple $\binom{u}{v} \in \Sigma^n \times \Sigma^m$

$$\begin{aligned} c(A \to \gamma) &:= \mathbb{E}\big[\#\text{rule applications } A \to \gamma\big] \\ &= \sum_{\substack{1 \leqslant i \leqslant j \leqslant n+1 \\ 1 \leqslant k \leqslant l \leqslant m+1}} \nu\left(\begin{smallmatrix} i & j \\ k & l \end{smallmatrix}, A \to \begin{smallmatrix} \gamma_1\bullet \\ \gamma_2\bullet \end{smallmatrix}\right) \cdot z\left(\begin{smallmatrix} i & j \\ k & l \end{smallmatrix}, A \to \begin{smallmatrix} \gamma_1\bullet \\ \gamma_2\bullet \end{smallmatrix}\right). \end{aligned}$$

Then we can take

$$\frac{c(A \to \gamma)}{\sum_{r \in R_A} c(r)}$$

as an estimate for the rule probability $P(A \to \gamma)$ that can be computed using only forward and reverse values of our items in the inside semiring.

3.6.3. Viterbi parses

Using the Viterbi semiring $([0,1], \max, \cdot, 0, 1)$ in a forward parser run yields a value for the goal item. This value is the probability of the most likely derivation of the input. To get the whole derivation *tree*, we do a backtracing step. The most likely derivation includes exactly the items that contributed the maximum[37] to the derived item value. Such a maximization step is only performed at a completion rule — only there do we have several item combinations to choose from.

We will not store pointers to items contributing maxima in the parser run — which is most time-efficient, but needs more memory. Instead, we start at the goal item and re-do the computations that determined its value. The pair of items yielding the maximal contribution are pushed on a stack and are then recursively handled the same way.

This backtrace step runs in $\mathcal{O}(r \cdot n)$ for SCFGs and $\mathcal{O}(r \cdot n \cdot m)$ for 2D-CFGs, where r is the number of rules in the most-likely derivation. As we do not allow left-recursion in the grammars, every derivation can use every rule at most once, before producing a new terminal at the left. So, r is at most $|R| \cdot n$ and $|R|(n+m)$ respectively.

[37] It may happen that there are several maximal contributions. In this case, we choose one at random.

3.7. Training with known structures

Although expectation maximization allows to train rule probabilities with primary structures only, better models result from training with known structures. Therefore, we provide our tool with means to include known secondary structures.

Conceptually, the case of known structures is much simpler than training with unknown structures: By assumption, there is exactly one (leftmost) derivation representing the (joint) secondary structure in question and the terminal word produced by this derivation is the according primary structure. Training of rule probabilities then simply means *counting* the number of rule applications for every rule and taking relative frequencies as the maximum likelihood estimate of rule probabilities.

However, known (joint) secondary structures are not given as a derivation tree — which is grammar-dependent and therefore not a suitable exchange format. Typically, an encoded form is used, somehow equivalent to the bar-bracket-notations introduced in sections 2.3 and 3.2, respectively.

With classical implementations of Earley parsers, simply using an unambiguous training grammar — which leaves the terminal placeholders `|`, `(`, `)`, `[` and `]` unexpanded — is quite efficient as significantly less items are derivable for that grammar. Our implementation, however, is optimized for *dense* grammars, sacrificing efficiency for unambiguous grammars: Our parser needs cubic time for *any* grammar.

The ideal solution clearly is to write a second, classical implementation of our parsers and use those for training with known secondary structures. Yet currently, only very few trusted RNA joint secondary structures are available in the literature. Therefore, it is feasible to simply use the same parser with an unambiguous training version of our grammars and compute the 'most likely' derivation — which of course is the *only* possible derivation in an unambiguous grammar. Then we will count rule occurrences in this derivation.

4. Implementation Design

In this chapter, we present the design of our implementation of the 2D-CFG Earley-style parser. As mentioned in section 3.5, we will use an ordinary SCFG Earley parser as a subroutine for performance improvements.

This leads to a doubling in the description of many aspects — everything exists in one- and two-dimensional versions. To remedy this redundancy, our discussion will always consider the 2D-CFG case. Only if simple deletion of indices for the second dimension or similarly simple modifications do not suffice, we will explicitly address SCFGs.

4.1. Motivating observations

As we have a specific application in mind for our parser — namely the prediction of joint secondary structures — we will tailor it to gain advantage of some properties of this task. So, before we discuss our design decisions, here are the observations underlying our design.

When looking at grammars for (joint) secondary structures, one notices that most substructures are possible in most locations of the primary structure — when totally ignoring bases, only the minimum size of hairpins is restrictive. This directly implies, that most grammar rules can be completed in almost all starting positions. We call such grammars **dense grammars**.

For our parser, this means that the vast *majority of items* is indeed *derivable* for any input word pair. Stated the other way round, we can exclude hardly any items up front.[38] This effect is amplified further by the use of rule templates (see section 3.1.5).

4.2. Item order

An item A **depends** on item B if the calculation of the (forward) value of A, $\nu(A)$, needs $\nu(B)$. The grammars our parser can handle will never yield loops in the item dependence relation. Therefore we can find a total order $\prec$ on the items, which implies that we can compute all item values in a dynamic programming scheme (see section 4.3).

Fortunately, it turns out that $\prec$ does *not* depend on the input words $\binom{u}{v}$ and can therefore be determined once and for all![39] Actually, once and for all *for each grammar* — we need to order the rules — but the process is quite simple and although we will do it manually here, it may be automated easily.

[38] In G_{RIP}, only the rules for G, H and J allow for 'up-front exclusion': They all produce an internal bond, hence, the endpoints of those must have distance of at least 4.

[39] As indicated in [Goo98, Goo99], this need not be true for every semiring parser and grammar.

4.2.1. Choice of rule indices ind(r)

For an mD-CFG $G = (N, \Sigma, R, S, P)$ — or a plain SCFG, as the definition works for those as well — we define the **left-corner relation** $\overset{\lessdot}{\Rightarrow}_G \subseteq N \times N$ by

$$A \overset{\lessdot}{\Rightarrow}_G B \qquad \text{iff} \qquad A \Rightarrow_G^+ B\gamma\,.$$

The name 'left-corner relation' is motivated from derivation trees: $A \overset{\lessdot}{\Rightarrow} B$ means that there is a partial derivation tree rooted at A, whose lower left corner is B. Our definition hides the different situations leading to $A \overset{\lessdot}{\Rightarrow} B$: The obvious case is a rule $A \to B\beta$. But there may also be rules $A \to EB\beta$ and $E \to \varepsilon$, leading to $A \overset{\lessdot}{\Rightarrow} B$, as well. And of course, we may need to chain several rules of both kinds to reach B.

Now we can define the rule indices *implicitly* — we require for rules $A \to \alpha$, $B \to \beta \in R$

$$A \overset{\lessdot}{\Rightarrow} B \qquad \text{implies} \qquad \mathrm{ind}(A \to \alpha) > \mathrm{ind}(B \to \beta)\,. \tag{4.1}$$

In general (mD-)CFGs, this may be unsatisfiable, namely, when there are $A, B \in N$ with both $A \overset{\lessdot}{\Rightarrow} B$ and $B \overset{\lessdot}{\Rightarrow} A$. But this implied (indirect) **left-recursion** in the grammar — $A \Rightarrow^* B\gamma \Rightarrow^* A\delta\gamma$ — and we excluded such grammars explicitly by requirement (6) in section 3.4. Therefore it is always possible — by topological sorting — to define rule indices $\mathrm{ind}(r)$ for every $r \in R$ fulfilling equation (4.1).

A sloppy interpretation of equation (4.1) might help motivating it: Given 'suitable' items[40] $I_1 := A \to \bullet B\alpha$ and $I_2 := B \to \beta\bullet$, we can conclude $I_3 := A \to B\bullet\alpha$ by completion; so I_3 depends on I_1 and I_2. Since I_2 and I_3 have identical indices, we have to put I_2 in front of I_3 by means of the rule index, and equation (4.1) does just that.

4.2.2. Definition of $\prec$

Now that we have the rule indices, we are ready to define $\prec$.

In section 4.6 on page 61, we will see that it suffices to consider items, where the dot sits in front of a non-terminal or at the right end. For those positions, the first and second dimension of q will always agree. Therefore we will later only have a one-dimensional dot index and we define the item order already for this optimization.

For an item $I := \left(\begin{smallmatrix} i & j \\ k & l \end{smallmatrix}, p, q\right)$ we define the **index vector**

$$o(I) := \begin{pmatrix} j \\ l \\ -i \\ -k \\ p \\ q \end{pmatrix} \tag{4.2}$$

and can now state $\prec$ as a total order by

$$I \prec I' \qquad \text{iff} \qquad o(I) \underset{\text{lex}}{<} o(I')\,,$$

where $\underset{\text{lex}}{<}$ represents the lexicographical order on vectors.

[40] Actually, these are only dotted rules. To explain the idea, indices are not necessary — given they match.

This means $\prec$ sorts the items

1. by j, ascending order
2. by l, ascending order
3. by i, *descending* order
4. by k, *descending* order
5. by p, ascending order
6. by q, ascending order

We did not discuss the rule existence items $\text{Rule}\left[A \to \gamma\right]$ so far. These items always have the value $v\big(\text{Rule}\left[A \to \gamma\right]\big) = P(A \to \gamma)$[41], therefore we simply put those *in front* of all 'real' items.[42]

In the following insertions, we will discuss what implications *would* be caused, if we *allowed* some variation to our parsing scenario. The complications arising there may make it plausible why we decided against these scenarios. To make perfectly clear that these are excursions into what we will *not* do, we present them 'bracketed out'.

How about 'real' Earley prediction?

Had we wanted to implement 'real prediction', i. e. only predict items according to the modified inference rule

$$\frac{\text{Rule}\left[B \to \binom{\beta_1}{\beta_2}\right]}{\binom{j\ \ j}{l\ \ l}, B \to \binom{\bullet\beta_1}{\bullet\beta_2}} \binom{i\ \ j}{k\ \ l}, A \to \binom{\alpha_1 \bullet B_1 \gamma_1}{\alpha_2 \bullet B_2 \gamma_2},$$

we would have had more trouble finding $\prec$.

Consider two rules $r_A = A \to B\alpha$ and $r_B = B \to \beta$, we have $A \overset{\leftarrow}{\Rightarrow} B$ and therefore — according to equation (4.1) — $p := \text{ind}(r_A) > \text{ind}(r_B) =: p'$.

Now consider some items using those rules:

$$I := \binom{j\ \ j}{l\ \ l}, A \to \binom{\bullet B_1 \alpha_1}{\bullet B_2 \alpha_2} \quad \text{and}$$
$$I' := \binom{j\ \ j}{l\ \ l}, B \to \binom{\bullet\beta_1}{\bullet\beta_2}.$$

Obviously, we would like to predict I' from I, i. e. use I as side condition in the modified prediction inference rule to derive I'. In $o(I)$ and $o(I')$ the first four components are equal, so the rule index decides about the order: $p > p'$ gives $I \succ I'$. Oops! That is just the wrong relation — we need $I \prec I'$.

In fact, concerning rule indices, completion and prediction are opposites:

	Completion	Prediction
Premise	$B \to \beta\bullet$	$A \to \bullet B\alpha$
Conclusion	$A \to B \bullet \alpha$	$B \to \bullet\beta$
needed order	$r_B \prec r_A$	$r_A \prec r_B$

[41] The rule probability is the item value of rule existence items in the inside semiring and the Viterbi semiring. For other semirings, other definitions might be needed.

[42] To be correct, we should also define an order among the rule existence items. It really does not matter, so let us simply take $\text{Rule}\left[A \to \alpha\right] \prec \text{Rule}\left[B \to \beta\right]$ iff $\text{ind}(A \to \alpha) < \text{ind}(B \to \beta)$.

The solution to this is to *reverse* the rule order for prediction. A clever definition of $\prec$ can do that ... but fortunately, we do not need that, so
Praise optimistic prediction!

How about left-recursion?

If we allowed left-recursion in the grammar, our definition of $\prec$ cannot work for *any* choice of rule indices: Consider the CFG

$$\begin{array}{lll} S \to A & & \\ A \to AaaB & A \to AaB & A \to \varepsilon \\ B \to a & B \to aa & \end{array}$$

We parse aaa by showing relevant items.

$$\begin{array}{lll} (1\ 1, A \to \bullet A) & & \\ (1\ 1, A \to \bullet AaB) =: I_1 & (1\ 1, A \to \bullet AaaB) =: I_2 & (1\ 1, A \to \bullet \varepsilon) \\ & & (1\ 1, A \to \varepsilon \bullet) \\ (1\ 1, A \to A \bullet aB) & (1\ 1, A \to A \bullet aaB) & \\ (1\ 2, A \to Aa \bullet B) & (1\ 3, A \to Aaa \bullet B) & \\ (2\ 2, B \to \bullet aa) & (3\ 3, B \to \bullet a) & \\ (2\ 4, B \to aa \bullet) & (3\ 4, B \to a \bullet) & \\ (1\ 4, A \to AaB \bullet) =: I_3 & (1\ 4, A \to AaaB \bullet) =: I_4 & \end{array}$$

In this situation, we can find two completion instances, namely for I_1, I_4 and for I_2, I_3, yielding

$$\frac{(1\ 1, A \to \bullet AaB) \quad (1\ 4, A \to AaaB \bullet)}{(1\ 4, A \to A \bullet aB)} \quad \text{and} \quad \frac{(1\ 1, A \to \bullet AaaB) \quad (1\ 4, A \to AaB \bullet)}{(1\ 4, A \to A \bullet aaB)}\,.$$

Here, the conclusion items and I_3 and I_4 respectively have the same position indices, so according to $\prec$, the rule index decides.

Let's call $r_1 := A \to AaB$ and $r_2 := A \to AaaB$. But then the first pair needs $\mathrm{ind}(r_2) < \mathrm{ind}(r_1)$ and the second one $\mathrm{ind}(r_1) < \mathrm{ind}(r_2)$. Hence those items can only be ordered by taking the dot position into account as well, which is not possible in our lexicographical order. ↯

The simple kind of order we use allows us to implement the iteration over all items via nested `for`-loops, which definitely contributes to the efficiency of our implementation.

4.2.3. Correctness of $\prec$

Now, we will argue that computing (forward) item values in order $\prec$ is possible. Semiring parsing tells us that item values are computed according to inference rules. So we simply have to look through all inference rules

$$\frac{A_1, \ldots, A_k}{B} ,$$

whether for each B, all possible combinations of premises $A_1, \ldots, A_k$ are smaller than B — in $\prec$, of course. So here it goes:

Scanning Scanning rules always increase the end index in one dimension, so the conclusion is greater than the premise in $\prec$.

Completion
$$\frac{\left(\begin{smallmatrix} i & r \\ k & s \end{smallmatrix}, A \to \begin{smallmatrix} \alpha_1 \bullet B_1 \gamma_1 \\ \alpha_2 \bullet B_2 \gamma_2 \end{smallmatrix}\right) \quad \left(\begin{smallmatrix} r & j \\ s & l \end{smallmatrix}, B \to \begin{smallmatrix} \beta_1 \bullet \\ \beta_2 \bullet \end{smallmatrix}\right)}{\left(\begin{smallmatrix} i & j \\ k & l \end{smallmatrix}, A \to \begin{smallmatrix} \alpha_1 B_1 \bullet \gamma_1 \\ \alpha_2 B_2 \bullet \gamma_2 \end{smallmatrix}\right)} \quad =: \frac{I_1 \; I_2}{I} .$$

Let us first consider items I_1 and I. We discriminate two cases

(1) $r < j \vee s < l$
This immediately implies $I_1 \prec I$ as we sort ascending in the first two components of $o(I_1)$ and $o(I)$.

(2) $r = j \wedge s = l$
In this case, the first *five* components of $o(I_1)$ and $o(I)$ are equal, but the last component of $o(I_1)$ — namely the dot index — is less by one, giving us $I_1 \prec I$.

The remaining items I_2 and I have identical end indices, for the start indices we again distinguish the two cases:

(1) $i < r \vee k < s$
Then $I_2 \prec I$, as sorting by i and k is done *descending*.

(2) $i = r \wedge k = s$
This case implies $\alpha \Rightarrow^* \left(\begin{smallmatrix} \varepsilon \\ \varepsilon \end{smallmatrix}\right)$ by theorem 4 applied to I_1 and consequently $A \Rightarrow \alpha B \gamma \Rightarrow^* B\gamma$ because of I. This means $A \overset{\leqslant}{\Rightarrow} B$, so by definition, we have $\mathrm{ind}(A \to \alpha B \gamma) > \mathrm{ind}(B \to \beta)$, and we have $I_2 \prec I$, as claimed.

Prediction
$$\frac{\mathrm{Rule}\left[B \to \left(\begin{smallmatrix} \beta_1 \\ \beta_2 \end{smallmatrix}\right)\right]}{\left(\begin{smallmatrix} j & j \\ l & l \end{smallmatrix}, B \to \begin{smallmatrix} \bullet \beta_1 \\ \bullet \beta_2 \end{smallmatrix}\right)} \quad =: \frac{I_1}{I} .$$

The special **rule existence items** $\mathrm{Rule}\left[B \to \beta\right]$ exist from the beginning and always have the constant value $P(B \to \beta)$, i. e. the rule probability. By definition of $\prec$, we have $I_1 \prec I$, so we are done.

□

4.3. Item representation

Since we have to perform quite a lot of computations on the item values, we need fast random access on them. Given the 'item density' reasoned about in the motivating section, it makes sense to save the item values in a large multi-dimensional *array* `V` that saves a value for *all* legal items. The array will be initialized with 0 — the semiring zero. 0 will be used to indicate the *absence* of an item. All computations done with item values of 0 behave exactly as if that item had not been derived.

As we will see in section 4.6, we only need to consider dot positions in front of non-terminals — or at the very right. Since all non-terminals have full dimension — (remember: $\binom{A}{B}$ is simply a shorthand for AB) — the dots in the first and second components always stay in sync, hence we only need to save *one* dot position `q`.

The array `V` will have six (!) indices, namely `i`, `j`, `k` and `l` — the position indices of the item — and `p` and `q` — the rule index and dot position. The order of the indices resembles the item order $\prec$:

$$\texttt{V}[\texttt{j},\texttt{l},\texttt{i},\texttt{k},\texttt{p},\texttt{q}] \quad \text{corresponds to} \quad v\left(\begin{smallmatrix}\texttt{i} & \texttt{j}\\ \texttt{k} & \texttt{l}\end{smallmatrix},\texttt{p},\texttt{q}\right).$$

Please pay attention to the order of the position indices — `j`, `l`, `i`, `k` — it differs from the intuitive one given in the items.

Analogously, we have an array `Z` that saves the reverse item values for applications that need a reverse parser run.

$$\texttt{Z}[\texttt{j},\texttt{l},\texttt{i},\texttt{k},\texttt{p},\texttt{q}] \quad \text{corresponds to} \quad z\left(\begin{smallmatrix}\texttt{i} & \texttt{j}\\ \texttt{k} & \texttt{l}\end{smallmatrix},\texttt{p},\texttt{q}\right).$$

Only `i`, `j`, `k` and `l` depend on the input words $\binom{u}{v}$, so the arrays have size in $\Theta(m^2n^2)$ — regarding the grammar as a constant. Considering the grammar as input, an additional factor *dr* shows up, for *dr* the number of dotted rules in the grammar.

We will use additional arrays for the preprocessing of one-dimensional subgrammars G_A for all non-terminals $A \in \mathcal{A}$:

$$\begin{aligned}\texttt{Vone}_A[\texttt{j},\texttt{i},\texttt{p},\texttt{q}] \quad &\text{corresponds to} \quad v(\texttt{i}\ \texttt{j},\texttt{p},\texttt{q}) \text{ in } G_A,\\ \texttt{Zone}_A[\texttt{j},\texttt{i},\texttt{p},\texttt{q}] \quad &\text{corresponds to} \quad z(\texttt{i}\ \texttt{j},\texttt{p},\texttt{q}) \text{ in } G_A.\end{aligned}$$

4.4. Optimistic Prediction

The prediction rule is

$$\frac{\text{Rule}\left[B \to \binom{\beta_1}{\beta_2}\right]}{\left(\begin{smallmatrix}\texttt{j} & \texttt{j}\\ \texttt{l} & \texttt{l}\end{smallmatrix}, B \to \begin{smallmatrix}\bullet\beta_1\\ \bullet\beta_2\end{smallmatrix}\right)}.$$

Since we expect all rules to be possible at almost every position, we simply *omit* the check whether an item exists that will use the newly predicted item later. Notice: The only thing this might cause is some unnecessary calculations — in no case does optimistic prediction harm correctness.

We will also profit from optimistic prediction in 1D-Earley. As already mentioned in section 3.5, we predict items $\left(\begin{smallmatrix} i & i \end{smallmatrix}, A \to \bullet\gamma\right)$ for every start position i and will complete all of those for every end position j — if such a completion is possible. This means, we need only one run of 1D-Earley per subgrammar, and can extract all needed information from it — we do *not* have to parse *every substring* separately.

4.5. Completion

Given the item order $\prec$, one simply has to iterate over all items in this order and compute their values as given in section 2.6 about semiring parsing. In fact, defining the item order $\prec$ as lexicographical order $\underset{\text{lex}}{\leqslant}$ on index vectors lets us implement the iteration via simple nested `for`-loops (see Algorithm A.1 on page 87 and A.7 on page 91).

In the completion rule — here it is again —

$$\frac{\left(\begin{smallmatrix} i & r \\ k & s \end{smallmatrix}, A \to \begin{smallmatrix} \alpha_1 \bullet B_1 \gamma_1 \\ \alpha_2 \bullet B_2 \gamma_2 \end{smallmatrix}\right) \quad \left(\begin{smallmatrix} r & j \\ s & l \end{smallmatrix}, B \to \begin{smallmatrix} \beta_1 \bullet \\ \beta_2 \bullet \end{smallmatrix}\right)}{\left(\begin{smallmatrix} i & j \\ k & l \end{smallmatrix}, A \to \begin{smallmatrix} \alpha_1 B_1 \bullet \gamma_1 \\ \alpha_2 B_2 \bullet \gamma_2 \end{smallmatrix}\right)} \quad =: \frac{I_1 \; I_2}{I} \,.$$

Fixing the conclusion item I, we know

- I_1's rule is the same as I's; the dot index is smaller by one.
- The start indices of I_1 are i and k.
- I_2's rule expands B and has the dots at the very right.
- The end indices of I_2 are j and l.

This leaves open

- the 'inner/split' indices r and s and
- the choice of a rule in R_B, the set of all rules expanding B.

So, we iterate over all r and s with $i \leqslant r \leqslant j$ and $k \leqslant s \leqslant l$ and over all rules $B \to \beta \in R_B$. For each such combination, we add its contribution to the value of the conclusion.

4.6. Immediate Scanning

The standard way of item value computation uses the inference rules *bottom-up*: The current item is plugged in as conclusion — if possible — and then we sum the contributions of all possible sets of premises. It is called bottom-up, since we know the bottom line, and look for a way up.

Scanning differs from completion in that there is always *at most one* premise item that can contribute to the conclusion's value. And the other way round: An item suitable for scanning — i. e. one with a terminal behind a dot — cannot be used in any other rule. Therefore, scanning may also be applied *top-down*: Once an item is produced[43], we can check whether it qualifies for scanning and if it does, *immediately* apply scanning.

[43] The first time the value of an item is increased, it leaves the state of non-existence (value 0).

As nobody will ever use the pre-scan item, we can — and will — go a step further: We will not even *store* a value for an item, that can be scanned. Instead, we immediately do the scan — and as scanning simply assigns the value of the premise to the conclusion — this means simply *jumping* over terminals in rules. By this trick, we can save 30 % of precious memory[44] for all those pre-scan item values we do not have to store now — at the price of making things a little more complex.

4.6.1. Getting rid of pre-scan items

We simply change the range of dot indices, such that dot positions with a terminal to the right totally vanish. We define the homomorphism *delT* by

$$delT(x) := \begin{cases} x & \text{if } x \text{ is a non-terminal component} \\ \varepsilon & \text{otherwise} \end{cases},$$

that simply deletes all terminals and set $\#\mathsf{NT}(w) := \left|delT(w)\right|$. Obviously, #NT simply counts the non-terminal components in w. Now, let $A \to \alpha \bullet \beta$ be a dotted rule. We call $q^{\mathrm{NT}} := \#\mathsf{NT}(\alpha_1)$ the **non-terminal dot index** or **NT dot index** of that item.

Notice that q^{NT} is well defined for SCFGs *and* 2D-CFGs: For SCFGs, we have $\alpha = \alpha_1$. As all non-terminals in 2D-CFGs have full dimension, the number of non-terminal components in α_1 and α_2 is equal in every dotted rule. Therefore, we will sloppily write $\#\mathsf{NT}(\alpha)$ instead of $\#\mathsf{NT}(\alpha_1)$ even for two-dimensional α.

Using NT dot indices makes it unnecessary to store two dot indices in two-dimensional items — we just argued that the NT dot index is *equal* for both components. Therefore we will have items

$$\left(\begin{smallmatrix} i & j \\ k & l \end{smallmatrix}, p, q^{\mathrm{NT}}\right) \qquad \text{instead of} \qquad \left(\begin{smallmatrix} i & j \\ k & l \end{smallmatrix}, p, \begin{smallmatrix} q_1 \\ q_2 \end{smallmatrix}\right).$$

We will have to iterate over all possible NT dot indices, so it is worth having a notation for their maximal value: For a rule index $p = \mathsf{ind}(A \to \gamma)$ we define

$$\ell(p) := \#\mathsf{NT}(\gamma).$$

Then, we always have $0 \leqslant q^{\mathrm{NT}} \leqslant \ell(p)$. Intuitively, an NT dot index q^{NT} means, the dot is sitting in front the $(q^{\mathrm{NT}} + 1)$st non-terminal, or at the right end in case $q^{\mathrm{NT}} = \ell(p)$.

4.6.2. Jumping over terminals

By using the NT dot indices instead of normal dot indices, we are sure never to stop in front of a terminal. Now the only thing left, is 'jumping' over terminals — remember: It is not enough to advance the *dot* indices, we also need to advance to *position* indices. In the times of normal dot indices, adding one to dot indices meant adding one to position indices. With NT dot indices, this does not hold any longer. Given a rule index $p = \mathsf{ind}(A \to \gamma)$ and NT dot index q^{NT} we define

[44] In G_{RIP}, roughly 30 % of all possible dot positions have a terminal right of the dot.

$$jump(p, q^{NT}) := \begin{pmatrix} h(\gamma_1, q^{NT}) - h(\gamma_1, q^{NT}-1) - 1 \\ h(\gamma_2, q^{NT}) - h(\gamma_2, q^{NT}-1) - 1 \end{pmatrix},$$

$$\text{with} \qquad h(w, q) := \begin{cases} \max\{i : \#NT(w_{1,i}) = q\} & \text{if } q \geqslant 0 \\ 1 & \text{if } q < 0 \end{cases}.$$

The helper function h converts NT dot indices to normal dot indices, the difference between two adjacent values gives the number of terminals in between.

Finally, whenever we 'create' a new item $I := \left(\begin{smallmatrix} i & j \\ k & l \end{smallmatrix}, p, q^{NT}\right)$ — via prediction or completion — we have to add the *jump* offset to the end indices. So we store the value for I — the one we have just computed — at $V[j', l', i, k, p, q^{NT}]$ instead of $V[j, l, i, k, p, q^{NT}]$ with $j' = j + jump_1(p, q^{NT})$ and $l' = l + jump_2(p, q^{NT})$.

However, we know that terminals occur only at the left and right end of right hand sides. Therefore $jump(p, q)$ is zero for all $0 < q < \ell(p)$. So will use the convenience notation

$$jumpLeft(p) := jump\big(p, 0\big) \quad \text{and}$$
$$jumpRight(p) := jump\big(p, \ell(p)\big)$$

instead of *jump*.

Now that we happily observed that we can skip scanning, a reminder might be adequate: We still need to check for matching terminals *somewhere*; and even more: As we use rule templates for rules differing only in the terminals produced, we need to know the terminals to assign the correct rule probability! Fortunately, all these issues are solved by **late item value computation**:

4.7. Late item value computation

Classical Earley semiring parsing uses the rule probability in the prediction step. This does not allow the use of rule templates whose probability depends on terminals found in later scanning steps, as the value of the predicted item is fixed long before the terminals at the right are known.

Therefore, we *postpone* taking rule probabilities into account until *finalization* — i. e. until the dot arrives at the very right. This may happen directly after prediction — if the item is a terminating rule — or otherwise after the the *last* completion. For terminating rules, their probability is known at time of prediction — we know the part of the input to match with the rule's terminals. So we give those items the appropriate value. For non-terminating rules, the value of a newly predicted item is simply set to 1. Then we have to check in every completion, whether the conclusion's dot has reached the right end — if it has, we multiply its new value by the then known rule probability.

It is important to note, that items with the dot *not* at the very right — i. e. exactly those items, where the rule probability is still missing — are *only* used in completions deriving an item with the same rule. If the shifted dot is still not at the right end, not including the rule probability is the correct thing to do. And if the dot has reached the right end, we explicitly include the probability now, thereby maintaining our invariant. The right

premise in the completion rule always is a finalized item, by induction *containing* the rule probability for 'its' rule. So, everything fits nicely.[45]

Late item value computation is *only* correct for **commutative semirings**. All we change is the order of some multiplications, the factors remain the same — but changing the order can already change the result in non-commutative semirings. Fortunately, we will only need the two semirings given in section 2.6, namely the Viterbi-semiring and the inside-semiring, both of which are commutative.

The nice consequence of late item value computation is, that it allows us to use our concept of rule templates.[46] At the time of completion of an item $I := \left(\begin{smallmatrix} i & j \\ k & l \end{smallmatrix}, B \to \begin{smallmatrix} \beta_1 \bullet \\ \beta_2 \bullet \end{smallmatrix}\right)$, we know exactly the subword it produces, namely $\left(\begin{smallmatrix} u_{i,j-1} \\ v_{k,l-1} \end{smallmatrix}\right)$. As we assumed in requirement (5) of section 3.4, terminals are only allowed at the left or right end of a rule; thereby, it suffices to have i, j, k and l to know the absolute indices — i. e. the indices in u and v, respectively — of all terminals in β. So we can simply multiply the current value of I — which does not consider the rule probability so far — by the appropriate rule instance probability.

Requirement (5) allows us to nail down our notation of **joint probabilities**. When we discussed it in section 3.1.4 for general grammars, we wrote $P(A \to \gamma \ \wedge \ a_1, \dots, a_d)$ for the probability of rule $P\big(A \to \gamma(a_1, \dots, a_d)\big)$, i. e. the rule instance of $A \to \gamma$ that results from inserting $a_1, \dots, a_d$ as terminals in γ. Now, we now that in our grammars, rules will have terminals only at the left and right end of the upper and lower component. For $p = \mathrm{ind}(A \to \gamma)$, we write

$$P\left(p \ \wedge \begin{smallmatrix} u_{i,j-1} \\ v_{k,l-1} \end{smallmatrix}\right)$$

for the joint probability of rule template p and the rule instance that matches with the derivation $A \Rightarrow \gamma \Rightarrow^* \left(\begin{smallmatrix} u_{i,j-1} \\ v_{k,l-1} \end{smallmatrix}\right)$.[47]

This new notation allows us to compactly give the factor, we have to include when completing item $I := \left(\begin{smallmatrix} i & j \\ k & l \end{smallmatrix}, B \to \begin{smallmatrix} \beta_1 \bullet \\ \beta_2 \bullet \end{smallmatrix}\right)$ in a finalization step:

$$P\left(\mathrm{ind}(B \to \beta) \ \wedge \begin{smallmatrix} u_{i,j-1} \\ v_{k,l-1} \end{smallmatrix}\right) .$$

Finally, we are able to formally define the semantics of V — when introducing it, we vaguely said $V[j, l, i, k, p, q]$ "corresponds to" $\nu\left(\begin{smallmatrix} i & j \\ k & l \end{smallmatrix}, p, q\right)$, now we know the reason for this and state

$$\begin{aligned} V[j, l, i, k, p, q] \otimes P\left(p \ \wedge \begin{smallmatrix} u_{i,j-1} \\ v_{k,l-1} \end{smallmatrix}\right) \quad &:= \quad \nu\left(\begin{smallmatrix} i & j \\ k & l \end{smallmatrix}, p, q\right) \qquad \text{if } q < \ell(p) \\ V[j, l, i, k, p, q] \quad &:= \quad \nu\left(\begin{smallmatrix} i & j \\ k & l \end{smallmatrix}, p, q\right) \qquad \text{if } q = \ell(p) \end{aligned} \tag{4.3}$$

Note that this weird definition does not make any statement about the value of V in the first case if $P\left(p \wedge \begin{smallmatrix} u_{i,j-1} \\ v_{k,l-1} \end{smallmatrix}\right) = 0$. As output, we will only use item values, where $q = \ell(p)$, in which case every thing is fine. For backward parsing, however, we should keep this in mind.

[45] Actually, this is not true for backward parsing. The problems arising there are resolved in section 4.9.

[46] It is also necessary for length-dependent SCFGs, as outlined in chapter 6.1.2.

[47] There is at most one instance that matches, as all instances have terminals at *exactly* the same positions. If no instance matches, that probability is 0.

4.8. Keeping probabilities in range — the 4-times-trick

Probabilities for Viterbi parses typically get *very* small, as they are a product of rule probabilities. Inside and outside probabilities are a little bit better — they also sum up some values — but still soon get very small. Symbolic calculation of the item values is prohibitively slow, so we need to work with floating point arithmetic. Some of the computations we perform are not prone to typical numerical errors, for instance, all of the following do not introduce errors at all:[48]

- repeated multiplication by `1.0`
- repeated addition of `0.0`
- multiplication by `0.0` (yields exactly `0.0` again)
- check for equality with `0.0` is exact

So, our main woe are so-called *underflows*, i. e. numbers too close to zero to be distinguishable from `0.0` in our number representation. The "standard solution" to this is taking *logarithms* of the probabilities. This keeps numbers in range, even for awfully small absolute values. We even get free candy: Multiplication is replaced by cheap addition ... but as typical for free candy, there is also a downside. Addition is not directly possible — simply taking exponentials then summing and taking logarithms again yields underflows. There are ways to avoid that, e. g. the formula

$$a = b + c \quad \Longleftrightarrow \quad a = b \cdot \left(1 + \frac{c}{b}\right) \quad \Longleftrightarrow \quad \log a = \log b + \log\Big(1 + \exp\big(\log c - \log b\big)\Big) \,.$$

If b and c are roughly the same size, not much precision is lost, but we need one exponentiation, one logarithm and three additions/subtractions. As the Viterbi semiring only needs multiplication and maximum, we will compute the Viterbi values in logarithmic scale.

Since we will have a lot of additions during the computation in the inside semiring, we will use another approach, making use of application knowledge: (Consistent) stochastic grammars induce a probability distribution on their language, i. e. $\sum_{w \in \mathcal{L}(G)} p(w) = 1$. Non-trivial grammars generate words of arbitrary length — otherwise, the language would be finite. Rewriting the above sum gives a convergent series

$$1 = \sum_{n \geqslant 0} \underbrace{\sum_{\substack{w \in \mathcal{L}(G): \\ |w| = n}} p(w)}_{=: W_n} = \sum_{n \geqslant 0} W_n \,.$$

We know that the language of a structure prediction grammar is Σ^+. For RNA, $|\Sigma| = 4$, so we have exactly 4^n words of length n. Let's assume, that all those 4^n words appear with the same probability p_n, then we have

$$\sum_{n \geqslant 0} 4^n p_n = 1 \qquad \text{which implies} \qquad p_n = o(4^{-n}) \,.$$

[48] This is true for floating point numbers conforming to the IEEE 754 standard. Other specifications/implementations may differ, but since the formers are so widely used, we only looked at those.

Of course, we will not really have the same probability for all primary structure of the same length, but still this shows that the there is a contribution of 4^{-n} to the probability of a word $w \in \Sigma^n$, that only compensates for the exponential growth in the number of possible primary structures. This contribution makes the probabilities decrease *exponentially* in the structure size causing us so much trouble with precision.

The logarithm-approach changes this exponential decrease in a linear decrease we can cope with. Our approach simply *factors out* the exponential contribution of 4^{-n}, so for every probability p dealing with terminal words of size n, we will actually save $4^n \cdot p$ instead of p.

$$\begin{aligned} I_1 &:= \left(i \;\; j \;,\; p \,,\, q\right) &&\rightsquigarrow \quad \texttt{Vone}_A\left[j, i, p, q\right] && \text{includes factor } 4^{j-i}\,, \\ I_2 &:= \left(\begin{smallmatrix} i & j \\ k & l \end{smallmatrix} \;,\; p \,,\, q\right) &&\rightsquigarrow \quad \texttt{V}\left[j, l, i, k, p, q\right] && \text{includes factor } 4^{j-i} \cdot 4^{l-k}\,. \end{aligned} \tag{4.4}$$

This keeps numbers in reasonable range and allows normal arithmetic. In the rest of this section, we discuss how to achieve this in our computations — providing a proof of (4.4) by induction on the fly.

However, a warning should be issued, because we are actually giving up *semiring independence* here. As the '4-times-trick' is so handy for us, we will accept this deficiency and silence our conscience by shouting out:

! The 4-times-trick only works for semirings, where $\otimes$ is the ordinary multiplication on reals / integers. **!**

Of course, this is fulfilled for all semirings we intend to use, namely the Viterbi-semiring and the inside semiring (as defined on page 21).

4.8.1. Prediction

Predicted items deal with no terminal strings, so we do not have to change anything.

4.8.2. Scanning

Since we use immediate scanning, we jump over terminals. However, scanning increases the length of the terminal word in question. So we multiply the value to be written at the position shifted by *jump* by 4^{jump}. For the 2D-case, we multiply by $4^{jump_1 + jump_2}$.

4.8.3. Completion

Although you might already know this one by heart, here is the completion rule again:

$$\frac{\left(\begin{smallmatrix} i & r \\ k & s \end{smallmatrix} \;,\; A \to \begin{smallmatrix} \alpha_1 \bullet B_1 \gamma_1 \\ \alpha_2 \bullet B_2 \gamma_2 \end{smallmatrix}\right) \quad \left(\begin{smallmatrix} r & j \\ s & l \end{smallmatrix} \;,\; B \to \begin{smallmatrix} \beta_1 \bullet \\ \beta_2 \bullet \end{smallmatrix}\right)}{\left(\begin{smallmatrix} i & j \\ k & l \end{smallmatrix} \;,\; A \to \begin{smallmatrix} \alpha_1 B_1 \bullet \gamma_1 \\ \alpha_2 B_2 \bullet \gamma_2 \end{smallmatrix}\right)} \quad =: \frac{I_1 \;\; I_2}{I}\,.$$

Given the saved values for I_1 and I_2 fulfill (4.4), we get:

$$\begin{aligned}\text{contribution added to } \nu(I) &= 4^{r-i} \cdot 4^{s-k} \cdot \nu(I_1) \quad \otimes \quad 4^{j-r} \cdot 4^{l-s} \cdot \nu(I_2) \\ &= 4^{r-i} \cdot 4^{s-k} \cdot \nu(I_1) \quad \cdot \quad 4^{j-r} \cdot 4^{l-s} \cdot \nu(I_2) \\ &= 4^{j-i} \cdot 4^{l-k} \cdot \Big(\nu(I_1) \ \cdot \ \nu(I_2)\Big) \\ &= 4^{j-i} \cdot 4^{l-k} \cdot \Big(\nu(I_1) \otimes \nu(I_2)\Big) \ .\end{aligned}$$

So we get $4^{j-i} \cdot 4^{l-k}$ in every contribution, giving us the correct value for I.

4.8.4. Finalization

Finalization does not change the length of terminal strings involved. The current value — by induction — includes the correct amount of fours, finalization simply multiplies this by the rule probability. □

4.9. Reverse parsing

Goodman describes in [Goo98, Goo99] how reverse semiring parsing generally looks like. From an abstract view, two things change: First, we need to reverse the item order — for our lexicographical order this means reversing the ordering of every component. Second, the formula for item value computations is a different one.

Not only is the reverse formula more complicated, it also reverses the typical inference rule application direction. In forward parsing it was bottom-up — the 'to-be-computed' item was the conclusion of a rule. Now it is top-down — the 'to-be-computed' item is one of the premises of a rule application.

We will essentially go through all topics of this chapter 4 and discuss what has to be done in reverse parsing.

4.9.1. Item order

The item order is the first thing to look at. We simply reverse the order defined in section 4.2, i. e. we order the items according to $\succ$ instead of $\prec$.

4.9.2. Item representation

As already indicated in section 4.3, we use a second array Z, that saves the reverse values, where

$$Z\big[j, l, i, k, p, q\big] \quad \text{corresponds to} \quad z\left(\begin{smallmatrix} i & j \\ k & l \end{smallmatrix}, p, q\right) .$$

Array Z is initialized to 0, except for the goal item:

$$Z\big[n+1, m+1, 1, 1, \mathrm{ind}(S' \to S), 1\big] := \mathbf{1} \ .$$

Again, the above "corresponds to"-phrase needs to be formalized. As we will see, for efficient computation we need to include the rule probability for the 'current' rule in Z,

although this probability is **not** part of the item value z as defined by GOODMAN in [Goo98]. The cause is late item value computation in the forward run. So we have

$$Z\big[j, l, i, k, p, q\big] \quad := \quad z\left(\begin{smallmatrix} i & j \\ k & l \end{smallmatrix}, p, q\right) \otimes P\Big(p \wedge \begin{smallmatrix} u_{i,j-1} \\ v_{k,l-1} \end{smallmatrix}\Big) \ . \tag{4.5}$$

Note, that for the backward values, we do not have a distinction of cases $q \not\leq \ell(p)$, so no weaseling out this time by "we only need finalized item values". Indeed, if $P\big(p \wedge \begin{smallmatrix} u_{i,j-1} \\ v_{k,l-1} \end{smallmatrix}\big) = 0$, so is $Z\big[j, l, i, k, p, \ell(p)\big]$ and there is no chance for removing $P\big(p \wedge \begin{smallmatrix} u_{i,j-1} \\ v_{k,l-1} \end{smallmatrix}\big)$ by dividing by it — so what to do for outside probability $\beta\big(A; i, j, k, l\big)$ for $A := \mathit{source}(p)$?

Actually, there are two cases: If the corresponding inside probability $\alpha\big(A; i, j, k, l\big)$ is non-zero, there must be a rule p' in R_A with $P\big(\mathrm{ind}(p') \wedge \begin{smallmatrix} u_{i,j-1} \\ v_{k,l-1} \end{smallmatrix}\big) > 0$. Then we can take the backward value $z\left(\begin{smallmatrix} i & j \\ k & l \end{smallmatrix}, p', \ell(p')\right)$ and divide it by $P\big(\mathrm{ind}(p') \wedge \begin{smallmatrix} u_{i,j-1} \\ v_{k,l-1} \end{smallmatrix}\big)$.

For the other case, $\alpha\big(A; i, j, k, l\big) = 0$, there is *no way* to efficiently recover the correct outside probability. But since all applications of inside and outside probabilities known to the author somehow *multiply corresponding* inside and outside probabilities, we simply set $\beta\big(A; i, j, k, l\big) := 0$ in such cases: Multiplying by corresponding inside probability will yield 0 anyway.

For the one-dimensional case, we use the arrays $\mathtt{Zone}_A$ with analogous semantics.

4.9.3. Computation of reverse values

A reverse parse will consist of a main loop over all legal items in reverse order. For each item, we compute its reverse value according to GOODMAN's formula:

$$z(A) = \bigoplus_{\substack{A_1, \ldots, A_k, j, B : \\ \frac{A_1, \ldots, A_k}{B} \ \wedge \ A_j = A}} z(B) \otimes \nu(A_1) \otimes \cdots \otimes \nu(A_{j-1}) \otimes \nu(A_{j+1}) \otimes \cdots \otimes \nu(A_k) \ .$$

Note that we need the forward values in arrays $\mathtt{V}$, $\mathtt{Vone}_u$ and $\mathtt{Vone}_v$, to compute reverse values.

Assume the current item is

$$I := \left(\begin{smallmatrix} i & j \\ k & l \end{smallmatrix}, A \to \begin{smallmatrix} \alpha_1 \bullet \beta_1 \\ \alpha_2 \bullet \beta_2 \end{smallmatrix}\right) .$$

There are several mutually exclusive cases for I and those cases determine how its reverse value has to be computed:

- The dot sits left of a **non-terminal**.
  Such items only occur as the *first* premise in the completion rule. We call value computations for such items **completion of first type**.
- The dot is at the very right — i. e. left of **nothing**.
  This means, I is a finalized item. Such items only occur as *second* premise in the completion rule. This case is handled in **completion of second type**.
- The dot is left of a **terminal**.
  Such items only occur as premise in **scanning** rules.

What about **prediction**?

The premise of prediction is a special rule existence item. We will not compute reverse values for those, as they do not have a meaningful application. So, no predictions in reverse parsing.

Such 'reverse predictions' would be somewhat like predicting the past and who would be interested in that?!

Section 3.6 on page 50 tells us that we only need *one kind* of reverse item values, namely those of items with the dot at the very right. This means:

There is no need to compute reverse values for items where
- the dot is *not* at the very right and
- there are no non-terminals *left of the dot* left.

4.9.4. Scanning

The box above leaves us with terminals at the right end of the rule. When we newly create a finalized item — i. e. one with the dot at the right end — we *cannot* immediately scan it like we did in the forward parse: We need the reverse value of the *not-scanned* item at the indices *including* terminals, because those correspond to the outside probabilities (for the inside semiring).

So, we have to make sure to jump over these terminals, whenever we *use* these reverse values later for further computations.

4.9.5. Completion — first type

Here is the completion rule — again — but with indices renamed to match I with the first premise:

$$\frac{\left(\begin{smallmatrix} i & j \\ k & l \end{smallmatrix}, A \to \begin{smallmatrix} \alpha_1 \bullet B_1 \gamma_1 \\ \alpha_2 \bullet B_2 \gamma_2 \end{smallmatrix}\right) \quad \left(\begin{smallmatrix} j & r \\ l & s \end{smallmatrix}, B \to \begin{smallmatrix} \beta_1 \bullet \\ \beta_2 \bullet \end{smallmatrix}\right)}{\left(\begin{smallmatrix} i & r \\ k & s \end{smallmatrix}, A \to \begin{smallmatrix} \alpha_1 B_1 \bullet \gamma_1 \\ \alpha_2 B_2 \bullet \gamma_2 \end{smallmatrix}\right)} \quad =: \frac{I \quad I_1}{I_2} \,.$$

We compute the reverse value of I by iterating over all possible choices for I_1 and I_2 — namely $j \leqslant r \leqslant n+1$, $l \leqslant s \leqslant m+1$ and all rules $B \to \beta \in R_B$ — and sum their contributions:

$$Z(I) \oplus= Z(I_2) \otimes V(I_1) \,.$$

Iterating in order $\succ$ makes sure that $Z(I_2)$ has already been computed.

4.9.6. Completion — second type

The renamed completion rule says:

$$\frac{\left(\begin{smallmatrix} r & i \\ s & k \end{smallmatrix}, A \to \begin{smallmatrix} \alpha_1 \bullet B_1 \gamma_1 \\ \alpha_2 \bullet B_2 \gamma_2 \end{smallmatrix}\right) \quad \left(\begin{smallmatrix} i & j \\ k & l \end{smallmatrix}, B \to \begin{smallmatrix} \beta_1 \bullet \\ \beta_2 \bullet \end{smallmatrix}\right)}{\left(\begin{smallmatrix} r & j \\ s & l \end{smallmatrix}, A \to \begin{smallmatrix} \alpha_1 B_1 \bullet \gamma_1 \\ \alpha_2 B_2 \bullet \gamma_2 \end{smallmatrix}\right)} \quad =: \frac{I_1 \quad I}{I_2} \,.$$

We compute the value by executing

$$Z(I) \oplus= Z(I_2) \otimes V(I_1)$$

for each $1 \leqslant r \leqslant i$, $1 \leqslant s \leqslant k$ and each dotted rule $A \to \alpha\bullet B\beta$. To conveniently do that, we use function *dottedRules* : $N \to \mathbb{N}^2$ that tells us for every non-terminal B, which combinations of rule and dot index produce a dotted rule $A \to \alpha\bullet B\beta$:

$$\mathit{dottedRules}(B) := \left\{ (p, q^{NT}) \;:\; p = \mathsf{ind}(A \to \alpha B\beta) \;\wedge\; q^{NT} = \#NT(\alpha) \right\}.$$

Additionally, the definition for Z in (4.5) tells us, that the backward value of the newly created finalized item I has to include the joint probability of $P\big(\mathsf{ind}(B \to \beta) \wedge \frac{u_{i,j-1}}{v_{k,l-1}}\big)$, so we multiply $Z(I)$ by that.

This has an additional advantage: Equation (4.3) did not properly define entries $V\big[j,l,i,k,p,q\big]$ for $q < \ell(p)$ and $P\big(p \wedge \frac{u_{i,j-1}}{v_{k,l-1}}\big) = 0$. In reverse parsing, we only use such entries in completion of second type, namely as $V(I_1)$. But for $Z(I)$, we always multiply $V(I_1)$ by $Z(I_2) = Z\big[r,s,j,l,p,q+1\big]$, by definition *including* $P\big(p \wedge \frac{u_{i,j-1}}{v_{k,l-1}}\big) = 0$. So, the undefined value of $V\big[j,l,i,k,p,q\big]$ is always 'zeroed' out when used.

4.9.7. 4-times-trick

In section 4.8, we argued that it makes sense not to directly save item values, but 4^n times that value — where n is the length of 'involved' terminal words. In subsection 3.6.1, we show that reverse item values in the inside semiring directly correspond to outside probabilities. So, looking at the terminal strings involved in outside probability

$$\beta(A,i,j) = \Pr\left[S \Rightarrow^* w_{1,i-1}\, A\, w_{j,n}\right]$$

tells us to multiply the reverse value of $\big(\begin{smallmatrix} i & j \end{smallmatrix}, A \to \gamma\bullet\big)$ by $4^{i-1} \cdot 4^{n-j+1} = 4^{n-(j-i)}$. Now, we can complete our table:

$$I_1 := \big(\begin{smallmatrix} i & j \end{smallmatrix}, p, q\big) \quad \rightsquigarrow \quad \begin{array}{ll} \mathtt{Vone}_A\big[j,i,p,q\big] & \text{includes factor } 4^{j-i}, \\ \mathtt{Zone}_A\big[j,i,p,q\big] & \text{includes factor } 4^{n-(j-i)}, \end{array}$$

$$I_2 := \left(\begin{smallmatrix} i & j \\ k & l \end{smallmatrix}, p, q\right) \quad \rightsquigarrow \quad \begin{array}{ll} V\big[j,l,i,k,p,q & \text{includes factor } 4^{j-i} \cdot 4^{l-k}, \\ Z\big[j,l,i,k,p,q\big] & \text{includes factor } 4^{n-(j-i)} \cdot 4^{m-(l-k)}. \end{array}$$

Simply inserting these for the reverse item value formulæ shows: We need to multiply by 4^s if we reverse-scan s terminal symbols. Then, both completion rules do the correct thing.

Note, that identity (3.3) including the correction factors becomes

$$4^{j-i} \cdot \nu(I) \cdot 4^{n-(j-i)} \cdot z(I) = \bigoplus_{\substack{D \text{ derivation}: \\ I \text{ appears in } D}} 4^n \cdot \nu(D)$$

for item $I = \big(\begin{smallmatrix} i & j \end{smallmatrix}, p, q\big)$ — which is good as $\nu(D)$ addresses the whole input $w \in \Sigma^n$, so 4^n is the correction factor we would like to see there and — even more important — 4^n does not depend on I.

5. Results

5.1. jackRIP — our C++ implementation of a 2D-CFG parser

Appendix A on page 85 contains pseudocode for the parser procedures described in the previous chapters. We also proudly present jackRIP, our implementation of these procedures in C++, providing a full-fledged command line interface for comfortable use of jackRIP.

Special emphasis was put on runtime efficiency. Apart from the measures taken in chapter 4, some technical optimizations have been incorporated. The most important is excessive use of precomputed look-up tables for functions listed in appendix A.1. Secondly, many functions were inlined, avoiding costs for subroutine calls. Last not least, the flyweight pattern was applied to create an object oriented design for terminals, nonterminals and rules with zero overhead compared to directly using integers as encoding.

5.2. Target Machine

Due to the high-dimensional arrays used in the parser, our main concern was to ensure that the whole array fits into main memory. We used a recent version (4.6) of the GNU C++-Compiler to cross-compile our parser for Windows 7. The target machine is comprised of four six-core AMD Opteron Processors, sharing 128 GB of main memory in a NUMA[49] architecture, running the 64-bit version of Windows 7.

5.3. Runtime efficiency tests

5.3.1. RNA-RNA-interaction

To the knowledge of the author, there is no software freely available which is close enough to our approach to do sensible runtime comparisons.

Of all approaches to the RNA-RNA interaction problem, the one by Kato et al. in [KAS09] is the most similar one, as it also uses a subclass of SMCFGs. Additionally, [KAS09, table 3] does not only show prediction accuracy measures, but also the CPU time used for computing the prediction. As the implementation was not available to the author, comparable values for the same machine could not be determined. It has also to be taken into account that the grammar used be Kato et al. is much smaller than G_{RIP}. Nevertheless does table 5.1 show the measures in comparison.

[49] **N**on-**U**niform **M**emory **A**rchitecture; this architecture distributes main memory to CPUs, but allows processes to access memory segments dedicated to a CPU other than the one they are currently running on. In the extreme this allows one CPU to use the whole main memory, providing us with the needed size for our parser.

RNA pair	n	m	runtime	memory	runtime from [KAS09]
DIS-DIS	35	35	180 s	290 MB	381 s
CopA-CopT	56	57	3,608 s	2.2 GB	1,206 s
ompA-MicA	137	72	51.7 h	17.7 GB	—
U2 and U6 snRNAs in yeast 21	144	95	147.6 h	33.7 GB	—

Table 5.1.: Comparison of runtime of jackRIP and the SRIG-parser from [KAS09]. Runtime and memory consumption for jackRIP were measured on above mentioned target machine (section 5.2). Note that the runtime shown in the last column refers to a different execution platform. The last two lines show the longest RNA pairs successfully predicted using jackRIP so far. The reference joint structures were taken from (top-down) [PSE+96], [WF02], [UDV+05] and [AZC05, id 22]

For these pairs $n \approx m$, so as both algorithms are known to have runtime in $\Theta(n^3m^3)$, we should expect rate of growth n^6. For jackRIP, the first two measures given show growth of $n^{6.4}$, which is within reasonable bounds of the expected growth. Taking into account the two large pairs, we see in the right plot of figure 5.1, that approximation $a \cdot n^6$ holds for larger inputs, as well.

However, taking the five values given in [KAS09, table 3] (left plot), we find that the polynomial rate of growth is definitely smaller than 6.

As mentioned below when discussing prediction quality, Kato et al. adapt their algorithm to the five input pairs used: They "set the probabilities of the rules of the type W at 0, that is, the algorithm does not calculate the recursion for this type of rule since we need not deal with bifurcation for the above test set" ([KAS09, section 6]). Completely removing the corresponding recursion in their algorithm would imply runtime in $\Theta(n^2m^2)$, still not fully explaining the runtime measures. So, further optimizations not discussed in the paper might have been applied to the SRIG-parser of Kato et al.

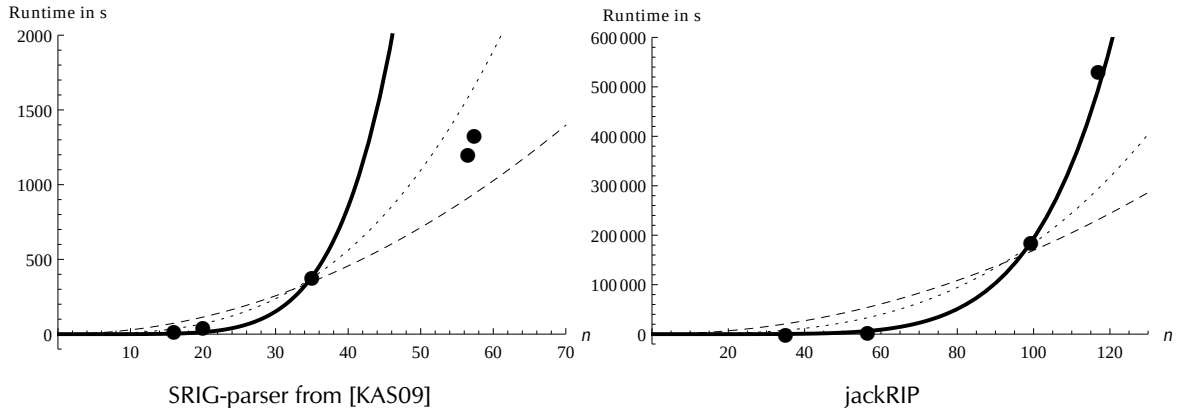


Figure 5.1.: "Rate of growth" analysis for the two parsers. The dots are the runtime measures, where the x-axis shows $\sqrt[6]{n^3m^3}$, which is n for $n = m$ and the y-axis indicates the runtime in seconds. The plotted functions were determined by least-square fitting of the *first three* data points on function $f(n) = a \cdot n^c$ for $c = 6$ (fat line), $c = 3$ (dotted line) and $c = 2$ (dashed line). The last two respectively one measure point(s) therefore allow to judge adequacy of the growth function.

Runtime and especially memory consumption of jackRIP get infeasible for RNA pairs with $n + m > 250$ on current machines — the same bound is given for the partition function approach based on rip2 in [HQRS10]. Table 5.1 gives the largest RNA pairs successfully predicted using jackRIP so far and might give an impression of what 'infeasible' means.

5.3.2. Comparison with classical Earley parsing

As no meaningful runtime comparison was possible for *joint* structure prediction, we looked at single RNAs: jackRIP is readily able to predict most likely secondary structures for single RNA molecules. An semiring implementation of the classical Earley-parser, using dynamically allocated lists as item sets, was compared to jackRIP's one-dimensional component.

For an RNA molecule of approximately 1000 bases, jackRIP needed 1 GB of memory and 35 minutes of time, whereas the classical Earley-parser had to be aborted after 20 hours, as it had eaten up over 50 GB of memory and was not making progress any more. This drastically shows the adequacy of using a full-storage array for all items in case of dense grammars.

5.4. Prediction quality tests

5.4.1. Test Data

Unfortunately, at the time of this writing only a few dozens of interacting RNAs are known, where the full joint secondary could be determined. Given that our approach needs to *train* a stochastic model — namely our grammar G_{RIP} — this poses quite a problem. Nevertheless did we set up two compilations of test data:

(1) A set of five interacting RNA pairs — all forming the same kind of kissing hairpin structure — compiled by Kato et al. in [KAS09].

(2) A set of 22 interacting RNAs used in [AZC05], which was kindly provided to us by Mirela Andronescu. Two pairs (namely the ones with ids 18 and 19) were excluded from the test set, since their known joint secondary structure contains internal pseudoknots.

Both sets were considered separately from each other and were divided into a *training set* and a *prediction set*.

5.4.2. Prediction method

The known structures from the training set were then used to give estimates of the rule probabilities by simply taking relative frequencies of rule counters. We started counting at one instead of zero to avoid completely ignored rules.

Then, a Viterbi parse was computed using the estimated probabilities for G_{RIP} and probabilities for G^{SecStr} determined from a training set of single RNAs containing a mixture of various types of RNA; namely the 'mixedRNAs' set used in [DE04], using parameters $m = 1$ and $k = 3$ for G^{SecStr}. The secondary structure encoded by this most likely parse tree was then output as prediction.

5.4.3. Quality measures

For evaluating the quality of predictions, we applied three simple measures typically used in secondary structure prediction:

Let P be the number of bonds in the predicted structure and R be the number of bonds in the trusted reference structure. Moreover, let M be the number of exact matches, i. e. the number of pairs identically occurring in predicted and reference structure. Then the following measures are defined

- **Sensitivity** $\frac{M}{R}$
- **Specificity** $\frac{M}{P}$
- **F-measure** $\frac{2M}{R+P}$

All measures are in range $[0, 1]$, larger values of course implying better prediction. Note: The f-measure is typically defined as $\frac{2\text{sensitivity}\cdot\text{specificity}}{\text{sensitivity}+\text{specificity}}$, the well-known formula for the harmonic mean of sensitivity and specificity. This is equivalent to our definition:

$$\frac{2\frac{M}{R}\cdot\frac{M}{P}}{\frac{M}{R}+\frac{M}{P}} = \frac{2\frac{M}{R}\cdot\frac{M}{P}}{M\cdot\frac{P+R}{R\cdot P}} = \frac{2M}{P+R}\,.$$

The reason, why the right form is preferred is simple: Sensitivity and specificity are not defined if R respectively P is 0. The formula typically given does not properly define the f-measure if *either* sensitivity *or* specificity is not defined, whereas ours still does.

In the case of *joint* secondary structures, 'bonds' is somewhat ambiguous. For comparability with other approaches that only predict interaction sites, we computed all three measures for

- the set of internal and external pairs, ignoring this difference and
- the set containing only the external bonds.

For the following tests, we always use the first variant.

5.5. Prediction results

5.5.1. Data from [KAS09]

	jackRIP			[KAS09], energy based model		
Structure	sensitivity	specificity	f-measure	sensitivity	specificity	f-measure
DIS-DIS	0.46	0.41	0.43	0.79	0.79	0.79
CopA-CopT	0.43	0.34	0.38	0.91	0.80	0.85

Table 5.2.: Comparison of prediction results for jackRIP and the energy based model from [KAS09].

Of the five joint secondary structures predicted in [KAS09], we used three structures to train G_{RIP}, namely Tar-Tar*, R1inv-R2inv and lncRNA54-RepZ. The other two ones were predicted using those probabilities. Table 5.2 on the preceding page shows the quality measures. In comparison, jackRIP performs rather poor, but several things need to be taken into consideration:

- The energy-based-approach used by Kato et al. does not suffer from little training data, since base pair stacking energies were used to determine rule probabilities, whereas jackRIP certainly does.[50]

  To make things worse, our grammar G_{RIP} is rather large in comparison, needing many rule probabilities to be trained from quite little data.

- In section 6 of [KAS09], the authors state that they "set the probabilities of the rules of the type W at 0, that is, the algorithm does not calculate the recursion for this type of rule since we need not deal with bifurcation for the above test set".

  Of course, this kind of manually tweaking gravely simplifies the task of prediction. The probabilities for jackRIP were *not edited in any way*, in order to measure performance for the more realistic setting of not knowing such details about the structure. Note that training G_{RIP} on a sufficiently large set of structures all forming this very same kind of kissing hairpins would automatically increase the probability of forming such structure for new RNAs.

5.5.2. Data from [AZC05]

Lack of training data is considered a main cause for mis-predictions in the first test above. Therefore, we added two more structures to the training set, see table 5.3 on the following page. G_{RIP} trained at this set, yielded the predictions shown in table 5.4 on page 77 for the rest of the test data. This table also lists the measures for predictions of pairFold, the implementation of the algorithms presented in [AZC05]. Looking at the results, several points are noteworthy:

- In the overall picture, jackRIP's predictions are slightly inferior to pairFold, but definitely comparable.

- One RNA pair drastically breaks the pattern: Tar-Tar*. This pair is the *only* pair (of which prediction data by both jackRIP and pairFold is available), where the concatenation of secondary structure as dot-bracket-word yields a pseudoknotted structure. More precisely, Tar-Tar* is known to form a kissing hairpin complex, correctly predicted by jackRIP. Since pairFold cannot predict such structures, it will have to approximate it by a well-nested structures — explaining its inferior prediction results.

- DIS-DIS was considered again, but this time using the new training data. The results significantly improve from the ones given in table 5.2, although the new training data

[50] Just to get an impression: For training, we count how often rules were applied in all training structures in total. For the three ones used here, the maximal count for a single rule was 43. Typical values of counters used for plain secondary structure prediction are hundreds or even thousands of applications.

contains a mixture of different RNA-RNA interacting pairs, whereas the result from table 5.2 used known structures very similar to the one of DIS-DIS.

This provides strong evidence that indeed too little training data was causing poor results in the first test, and on the other hand suggests the possibility of further improving predictions by choosing a larger set of pairs forming kissing loops.

Finally, note that our training set still consisted of only *12* RNA pairs, quite little data compared to studies for plain secondary structure prediction. Therefore, further improvements could be achievable by increasing training sets.

RNA pair	n	m	Origin
Hammerhead ribozyme R32 and substrate	11	32	[AZC05], id 01
Hammerhead ribozyme and no-tail gene target mRNA in zebrafish	17	38	[AZC05], id 03
αYRz60 hammerhead ribozyme and HIV-1 target sequence	70	100	[AZC05], id 05
Conventional hairpin ribozyme, seq. variation HP-WTSV1 and substrate	14	55	[AZC05], id 07
Hairpin ribozyme and substrate	21	92	[AZC05], id 09
Hairpin ribozyme RzG101 and substrate	50	14	[AZC05], id 11
X-motif ribozyme model 43X and S21 substrate RNA	21	43	[AZC05], id 13
ATP-sensitive allosteric ribozyme construct IV-up and substrate	14	59	[AZC05], id 15
5'CYbUT RNA crosslinked to gCYb-558	28	59	[AZC05], id 17
U2 and U6 snRNAs in yeast 21	72	40	[AZC05], id 21
fhlA-OxyS	113	109	[AA00, fig 7]
sodB-ryhB	87	90	[GT04, fig 7]

Table 5.3.: The RNA pairs whose known joint secondary structure is used for training G_{RIP}.

5.5.3. Summary

Both tests show the applicability of our method, although on average, previously published approaches could not be outperformed. On the other hand, those approaches did not rely on thoroughly training a stochastic model. Therefore a larger training set for jackRIP is very likely to improve upon prediction results.

RNA pair	Origin	jackRIP			[AZC05]		
		sens.	spec.	f-meas.	sens.	spec.	f-meas.
AUG-cleaving hammerhead-like ribozyme and substrate	[AZC05], id 02	0.79	0.94	0.86	0.95	0.90	0.92
Rz12×12 hammerhead ribozyme and HIV-1 target sequence	[AZC05], id 04	0.88	0.97	0.92	0.88	1.00	0.94
Reverse-joined hairpin ribozyme HP-RJTL and substrate	[AZC05], id 06	0.26	1.00	0.42	1.00	0.76	0.86
Hairpin-derived twin ribozyme HP-DS1 and substrate	[AZC05], id 08	0.63	0.87	0.73	0.93	0.78	0.85
Modified hairpin ribozyme and substrate	[AZC05], id 10	0.67	0.97	0.79	0.84	0.75	0.79
Minimal two-way helical junction 2WJ-SV5 hairpin ribozyme–substrate complex	[AZC05], id 12	0.50	1.00	0.67	0.92	0.81	0.86
MR8-1 ribozyme (derived from X-motif ribozyme) and S21 substrate RNA	[AZC05], id 14	0.29	0.45	0.36	0.71	0.48	0.57
ATP-insensitive ribozyme construct IV-down and substrate	[AZC05], id 16	0.83	0.95	0.88	0.83	0.70	0.76
Tar-Tar* (Kissing hairpin loop complex)	[AZC05], id 20	1.00	0.88	0.93	0.36	0.56	0.44
U4 and U6 snRNAs in yeast	[AZC05], id 22	0.65	0.71	0.68	0.94	0.68	0.79
ompA-MicA	[UDV$^+$05, fig 7]	0.48	0.66	0.56	—	—	—
DIS-DIS	[PSE$^+$96, fig 1]	0.50	0.64	0.56	—	—	—

Table 5.4.: Prediction quality measures for the predictions of jackRIP and pairFold as given in [AZC05].

6. Conclusion

In the previous chapters, we introduced the class of m-dimensional context-free grammars (mD-CFGs) and showed that many concepts of ordinary CFGs can easily be generalized. For $m = 2$, i. e. for the 2D-CFGs, we presented an Earley-style parser with worst case run time in $\Theta(n^3 m^3)$ and memory consumption in $\Theta(n^2 m^2)$ for a terminal 2-tuple $\binom{u}{v}$ of size $|u| = n$ and $|v| = m$. The parser is given in an item-based form suitable for semiring parsing — i. e. it allows probabilistic parsing. Note that no normal form for grammars is needed in Earley parsing.

We translated a partitioning scheme for RNA-RNA joint secondary structures into the 2D-CFG G_{RIP}, which can be used in generalizations of the known algorithms for RNA secondary structure prediction based on trained stochastic context-free grammars (i. e. inside and outside algorithm and computation of Viterbi parses). It should be noted, that internal and external pseudoknots, as well as so-called zig-zags are excluded from the allowed joint structures; however, kissing hairpin complexes are predictable.

For efficient implementation, we discussed some optimizations of Earley parsing for *dense* grammars, i. e. grammars, where every non-terminal can derive almost every possible substring of a terminal word. This is typically the case for structure prediction grammars, as the terminal strings are the primary structures. The optimizations do not change the $\mathcal{O}$-class of runtime or memory consumption, but conceptually convert Earley parsing into a dynamic programming algorithm of very regular shape. This allows modern CPUs to make heavy use of caching and pipelining. jackRIP, our implementation of the probabilistic parser for 2D-CFGs shows a significant speedup in practice compared to classical Earley parsing.

Finally, we compared the prediction results of jackRIP to other recent approaches for determining RNA-RNA interactions, showing that currently, prediction accuracy is slightly inferior. However, those approaches are based on energy-minimization, whereas jackRIP requires training of a stochstic model from known joint structures. At the time of this writing, only a very few trusted joint structures were known and experiments with varying amounts of training data provided strong evidence that lack of training data is a main reason for poor predictions. So, we hope for improving our results, as more joint structures become available.

6.1. Future Work

Although this work is quite self-contained, the algorithm and its implementation jackRIP, as well, have carefully been designed with some extensions in mind. In this section, we will briefly introduce those, including the changes needed for implementing them. Both concepts have not been applied to RNA-RNA interaction prediction so far.

6.1.1. Statistical Sampling based on trained models

Predicting a single most likely secondary structure is an intrinsically limited approach. It is well accepted in the community that perfectly reliable structure predictions are out of reach. Too many aspects involved in RNA folding in nature are overlooked even by the most accurate models known and stochastically trained models will always remain an approximation.

Therefore the idea evolved to compute, e. g. base pairing *probabilities* or even probabilities for the occurrence of a given loop type at a given position, thereby allowing to give 'classes' of probable structures instead of a single static prediction. All of this can be done using the partition functions introduced by McCaskill in [McC90]. Statistical sampling via partition functions has been generalized to RIP in [HQRS09].

Recent approaches in [NS10] show, how to compute the same information from carefully chosen SCFGs, which is very useful as trained SCFG-models are not suffering from deficiencies of energy models. All that is needed as input to this new sampling method is the set of inside and outside probabilities for the primary structure in question, which jackRIP can efficiently compute.

However, there is one pitfall we created by our decision in subsection 3.5.1: The outside probabilities in the subgrammars are defined to 'start' at the start symbol of the subgrammar — instead of the one of the main grammar. For sampling, this is the wrong probability! To repair this, we have to change the one-dimensional reverse parser, such that it does *not* start by setting the reverse value of the subgrammar's goal item to $\mathbf{1}$ and all others to $\mathbf{0}$, but by using "start values" computed by the two-dimensional reverse parser.

To fill in these start values, 2D-Earley-Backward needs a new subroutine 2D-Complete-Second-Type-1D-NT which computes the backward value of finalized 1D-items for the start rules of all one-dimensional subgrammars. These values would then be given as initialization of `Zone` to 1D-Earley-Backward.

6.1.2. Length-Dependency

Recently, an new extension to the formalism of SCFGs was proposed: Nebel and Weinberg introduce in [WN10] the class of **length-dependent** SCFGs, which differs from SCFGs in allowing a second parameter for determining the probability of a rule application: the length of the subword finally derived from this rule.

From the point of view of bioinformatics, this class is interesting, as it allows to assign probabilities for a specific loop structure, which depends on the *size* of this loop. Plain SCFGs can model such dependency to a certain extent, but are somewhat limited in the kind of functions in the length they realize:

Consider for example a non-terminal C and rules $C \to p : \mathtt{|}C$ and $C \to 1-p : \mathtt{|}$. Generating $\mathtt{|}^n$ from C has probability $p^{n-1}(1-p)$, which is of course a function in n; however if we need to have a function other than this geometric distribution, we need to cleverly modify our grammar, and most distributions will not be compactly representable. The new concept of length-dependency embraces all possible distributions.

For implementing an Earley parser for length-dependent grammars, one needs to postpone inclusion of rule probabilities to the completion step — only then is the length of the produced subword known. 'Luckily', this is exactly the way we do it in our [2D-/S]CFG parser. Hence, the only change needed to implement length-dependency concerns the computation of joint rule probabilities — 'joint' now embraces the rule template, produced terminals *and* produced length.

These new length-dependent grammars have not been applied to RIP yet. However, increasing the number of parameters to train needs more training data … lack of which we already had for plain grammars.

Appendix

A. Pseudocode

In this section, we will give pseudocode for the parsers implementing the design given in chapter 4. Assumptions made are:

- $G = (N, \Sigma, R, S', P)$ is a 2D-CFG, where one-dimensional subgrammars have been removed: $N = N_2 \dot\cup N_u \dot\cup N_v$, where the (effectively) two-dimensional non-terminals belong to N_2 and (effectively) one-dimensional ones to N_u and N_v — non-terminals with effective indices $\{1\}$ and $\{2\}$, respectively. In R, there are only rules expanding non-terminals in N_2.

  For every $A \in N_u \dot\cup N_v$ we have the subgrammar $G_A = (N_A, \Sigma, R_A, A, P_A)$ as defined in subsection 3.1.3. G_A is a SCFG.

- We are parsing $\binom{u}{v}$ with $u \in \Sigma^n$ and $v \in \Sigma^m$ in G.

- There is only one rule for S', namely $S' \to S$, $S \in N$ and
  for each G_A there is only one rule $A \to C$ for some $C \in N_A$.
  S' and C do not show up in right hand sides in R and R_A, respectively.

- G does not contain non-terminals of effective dimension zero.
  (Such non-terminals can simply be deleted.)

- Terminals do not occur in the middle of a rule.

- No left-recursion, i. e. $\nexists A \;\; A \Rightarrow^+ A\alpha$.

- We use a semiring where $\otimes$ is the multiplication on reals/integers.
  (Otherwise, the 4-times-trick does not work. In those cases you have to delete all multiplications with powers of four from the following code.)

The correctness of the algorithms follows from the proofs in [Goo98] and our discussions in chapters 3 and 4.

A.1. Collected definitions

During our refinement, we defined several functions that depend on the grammar, but not on the input $\binom{u}{v}$. As those values are used very often, we will precompute them and use table-lookup functions. In the following, we list them again for reference:

- $\mathrm{ind}(r)$ for $r \in R$:
  The rules of the grammar have to be sorted — by assigning them consecutive indices, such that $A \overset{\lessdot}{\Rightarrow}^+ B$ implies $\mathrm{ind}(A \to \alpha) > \mathrm{ind}(B \to \beta)$.

- $\mathrm{ind's}(B)$ for $B \in N$:
  The set of all rule indices whose left hand side is B, i. e. $\mathrm{ind's}(B) := \{\mathrm{ind}(r) : r \in R_B\}$

- $\mathit{source}(p) = B \;:\Leftrightarrow\; p \in \mathrm{ind's}(B)$, the left hand side of a rule.
- $\#\mathrm{NT}(\gamma)$, the number of non-terminal (components) in γ.
  Formally, define $\#\mathrm{NT}(\gamma) := \big|\mathit{delT}(\gamma)\big|$,
  for homomorphism delT given by $\mathit{delT}(x) := \begin{cases} x & \text{if } x \text{ is a non-terminal (component)} \\ \varepsilon & \text{otherwise} \end{cases}$.
- $\ell(p)$ for rule index $p = \mathrm{ind}(A \to \alpha)$, the number of non-terminals in α,
  i. e. $\ell(p) := \#\mathrm{NT}(\alpha)$,
- q^{NT}, the NT dot index, satisfies $0 \leqslant q^{\mathrm{NT}} \leqslant \ell(p)$, meaning
  the dot is sitting directly *left of* the $(q^{\mathrm{NT}}+1)$st non-terminal for $0 \leqslant q^{\mathrm{NT}} < \ell(p)$ or
  at the very *right end* in case $q^{\mathrm{NT}} = \ell(p)$
- $\mathit{dottedRules}(B) := \big\{(p, q^{\mathrm{NT}}) \;:\; p = \mathrm{ind}(A \to \alpha B \beta) \;\wedge\; q^{\mathrm{NT}} = \#\mathrm{NT}(\alpha)\big\}$,
  the indices of all dotted rules, where B is the next non-terminal.
  For subgrammar G_A, we write
  $$\mathit{dottedRules}_A(B) := \Big\{(p, q^{\mathrm{NT}}) \;:\; C \in N_A \;\wedge\; p = \mathrm{ind}(C \to \alpha B \beta) \;\wedge\; q^{\mathrm{NT}} = \#\mathrm{NT}(\alpha)\Big\},$$
  to restrict the allowed rules to those expanding non-terminals in G_A.
- $\mathrm{nextNT}\big(p, q^{\mathrm{NT}}\big)$ for rule index p and NT dot index $q^{\mathrm{NT}} < \ell(p)$:
  The $(q^{\mathrm{NT}}+1)$st non-terminal in right hand side of rule with index p,
  equivalently: $B = \mathrm{nextNT}(p, q) \iff (p, q) \in \mathit{dottedRules}(B)$
- $\mathit{jumpLeft}(p)$ and $\mathit{jumpRight}(p)$ for rule index p:
  The number of terminals at the left and right end of rule p, respectively.
- $P(p \;\wedge w)$ the joint probability for rule template p and terminals w:
  Let $p = \mathrm{ind}(A \to t^x \gamma\, t^y)$ for t the terminal placeholder, $x, y \in \mathbb{N}_0$ and $\gamma \in N^\star$.
  Then we have
  $$P(p \;\wedge w) := \Pr\left[A \to w_{1,x}\; \gamma\; w_{|w|-y+1,|w|}\right] \qquad \text{if } x + y \leqslant |w|\,.$$
  Generalization to two-dimensional w are straight-forward.

A.2. Invariants

The following invariants give the relation between the item values $v(I)$ and $z(I)$ defined by Goodman and the values we store in the arrays `V`, `Vone`, `Z` and `Zone`:

$$\begin{aligned}
\texttt{Vone}\big[j, i, p, q\big] \otimes P\big(p \;\wedge w_{i,j-1}\big) &= 4^{j-i} \cdot v\big(\begin{smallmatrix} i & j \end{smallmatrix}, p, q\big) && \text{if } q < \ell(p)\,,\\
\texttt{Vone}\big[j, i, p, q\big] &= 4^{j-i} \cdot v\big(\begin{smallmatrix} i & j \end{smallmatrix}, p, q\big) && \text{if } q = \ell(p)\,,\\
\texttt{V}\big[j, l, i, k, p, q\big] \otimes P\Big(p \;\wedge \begin{smallmatrix} u_{i,j-1} \\ v_{k,l-1} \end{smallmatrix}\Big) &= 4^{j-i} \cdot 4^{l-k} \cdot v\left(\begin{smallmatrix} i & j \\ k & l \end{smallmatrix}, p, q\right) && \text{if } q < \ell(p)\,,\\
\texttt{V}\big[j, l, i, k, p, q\big] &= 4^{j-i} \cdot 4^{l-k} \cdot v\left(\begin{smallmatrix} i & j \\ k & l \end{smallmatrix}, p, q\right) && \text{if } q = \ell(p)\,,\\
\texttt{Z}\big[j, i, p, q\big] &= 4^{n-(j-i)} \cdot z\big(\begin{smallmatrix} i & j \end{smallmatrix}, p, q\big) \otimes P\big(p \;\wedge w_{i,j-1}\big)\,,\\
\texttt{Z}\big[j, l, i, k, p, q\big] &= 4^{n-(j-i)} \cdot 4^{m-(l-k)} \cdot \\
&\qquad z\left(\begin{smallmatrix} i & j \\ k & l \end{smallmatrix}, p, q\right) \otimes P\Big(p \;\wedge \begin{smallmatrix} u_{i,j-1} \\ v_{k,l-1} \end{smallmatrix}\Big)\,.
\end{aligned}$$

A.3. 1D-Earley-Forward

The main procedure simply loops over all item positions in $\prec$ order. For the current item $I := (i \;\; j, p, q)$, two cases are distinguished directly in the main procedure: If $i = j$ the item is a prediction item, and 1D-Predict-Forward is called. Notice, that predict items *always* have dot index $q = 0$. Therefore we do not need a loop for q there. Otherwise, the item is a completion item and it is 1D-Complete-Forward's turn.

Strictly speaking, there are further cases checked for in those sub-procedures, namely whether the current item is a finalize item ($q = \ell(p)$). If that is the case, we have to include the rule probability.

The 4-times-trick is implemented as well: During prediction or for finalized items in completion, appropriate powers of 4 are included in the item values.

Algorithm A.1 The main loop over all items in $\prec$ order.

1D-Earley-Forward(A, w)

```
    // A is the start symbol of the subgrammar G_A to use.
    // w ∈ {u, v} is the terminal word to parse in grammar G_A.
    // Item values are stored in array Vone_A.
 1  n := |w|
 2  Init array Vone_A to 0                  // semiring zero
 3  for j := 1 to n + 1
 4      for p := 1 to |R_A|                 // iterate over rules
 5          1D-Predict-Forward(j, p)
 6      end for
 7      for i := j downto 1
            // for each dotted rule, do complete
 8          for p := 1 to |R_A|             // for all rules
 9              for q := 1 to ℓ(p)          // Start at 1: omit leftmost dot pos.
10                  1D-Complete-Forward(i j, p, q)
11              end for q
12          end for p
13      end for i
14  end for j
```

Algorithm A.2 Handling of prediction items.

1D-Predict-Forward(j, p)

```
    // Predict item (j  j , p , 0).
1   jump := jumpLeft(p)                          // immediate scan
2   if ℓ(p) == 0                                 // p is a terminating rule
3       prob := P(p ∧ w_{j,j+jump−1})            // determine rule probability
4       Vone_A[j + jump, j, p, 0] := prob · 4^jump   // Set value including 4s for skipped terminals
5   else
6       Vone_A[j + jump, j, p, 0] := 1 · 4^jump
7   end if
```

Algorithm A.3 Handling of completion items.

1D-Complete-Forward$(i\ \ j\ ,\ p\ ,\ q)$

```
    // Asserts 1 ⩽ q ⩽ ℓ(p), i. e. q may not be 0.
    // Asserts that all needed item values are computed.
 1  B := nextNT(p, q − 1)                        // Get the non-terminal that was completed
 2  if q == ℓ(p) // finalization
        // We have to scan the terminals at right end of rule
 3      jump := jumpRight(p)                     // The offset for writing
 4      prob := P(p ∧ w_{i,j+jump−1})            // rule probability
 5  else // no finalization
 6      jump := 0
 7      prob := 1
 8  end if
 9  for r := i to j
10      for each p_B ∈ ind's(B)                  // rules with left hand side B in G_A
11          Vone_A[j + jump, i, p, q] ⊕= Vone_A[r, i, p, q − 1] ⊗ Vone_A[j, r, p_B, ℓ(p_B)]
12      end for B-rule
13  end for r
    // Value complete, include rule probability now
14  Vone_A[j + jump, i, p, q] ⊗= prob · 4^jump
```

A.4. 1D-Earley-Reverse

The main reverse procedure loops over all items in reverse $\prec$ order. For a current item $I := (i \quad j, p, q)$, only the cases of first type and second type completion remain (see section 4.9). As indicated in the framed box on page 69, we can stop reverse parsing if the dot is not at the right end and no non-terminals are located left of the dot.

For fulfilling the invariants, we have to include the rule probability for the current rule in 1D-Complete-Second-Type.

Algorithm A.4 Main loop in reverse parse, over items in $\succ$ order.

1D-Earley-Backward(A, w)

```
// A is the start symbol of the subgrammar G_A to use.
// A → C is the only rule expanding A.
// w ∈ {u, v} is the input word to parse in grammar G_A.
// Computed values are stored in Zone_A.
// Assumes Vone_A is filled with forward values.
// Computes reverse values inside of subgrammar G_A. See section 3.5.1 for details.
 1  n := |w|
 2  Init array Zone_A to 0
 3  Zone_A[n + 1, 1, ind(A → C), 1] := 1
 4  for j := n + 1 downto 1
 5      for i := 1 to j
 6          for p := |R| downto 1
 7              1D-Complete-Second-Type(i, j, p)
 8              for q := ℓ(p) − 1 downto 1    // ⇝ if ℓ(p) ⩽ 1, skip loop
 9                  1D-Complete-First-Type(i  j, p, q)
10              end for q
11          end for p
12      end for i
13  end for j
```

Algorithm A.5 Handling of completion, second type.

1D-Complete-Second-Type(i, j, p_B)

```
    // Creates item (i  j , p_B , ℓ(p_B)).
 1  B := source(p_B)                  // B is left side of rule with index p_B. By assumption, B ∈ N_A.
 2  prob := P(p_B ∧ w_{i,j−1})        // Include the probability for rule p_B
 3  for r := 1 to i
 4      for each (p, q) ∈ dottedRules_A(B)        // the (q+1)st non-terminal in rule p is B
            // maybe we have to skip terminals at right end
 5          if q == ℓ(p) − 1
 6              jump := jumpRight(p)
 7          else
 8              jump := 0
 9          end if
10          c := 4^jump
            // Looking at items I_1 = (r  i , p , q) and I_2 = (r  j + jump , p , q + 1).
11          Zone_A[j, i, p_B, ℓ(p_B)] ⊕=
                    c · Zone_A[j + jump, r, p, q + 1] ⊗ Vone_A[i, r, p, q] ⊗ prob
12      end for (p, q)
13  end for r
```

Algorithm A.6 Handling of completion, first type.

1D-Complete-First-Type$(i\ \ j\ ,\ p\ ,\ q)$

```
    // Creates item (i  j , p , q).
    // Assumes, 0 < q < ℓ(p).
 1  B := nextNT(p, q)        // B is (effectively) one-dimensional for sure
 2  if q == ℓ(p) − 1         // maybe we have to skip terminals at right end of p
 3      jump := jumpRight(p)
 4  else
 5      jump := 0
 6  end if
 7  c := 4^jump
 8  for r := j to n + 1
        // Fixing I_2 = (i  r + jump , p , q + 1).
 9      for each p_B ∈ ind's(B) // rules with left hand side B in G_B
            // Fixing I_1 = (j  r , p_B , ℓ(p_B)).
10          Zone_A[j, i, p, q] ⊕=
                    c · Zone_A[r + jump, i, p, q + 1] ⊗ Vone_A[r, j, p_B, ℓ(p_B)]
11      end for p_B
12  end for r
```

A.5. 2D-Earley-Forward

Compared to the one-dimensional case, we have to make changes in two places: First, we call 1D-Earley-Forward for all subgrammars. This assures, that the forward values for one-dimensional items are precomputed in the arrays $\texttt{Vone}_A$. Second, in 2D-Complete-Forward we have to discriminate between possible cases for the effective indices of the currently completed non-terminal B.

The rest of the implementation is a straight-forward generalization of the 1D-case.

Algorithm A.7 The main loop over all items in $\prec$ order.

2D-Earley-Forward(u, v)

```
    // u and v are the two components of the input pair.
 1  n := |u|   ,   m := |v|
 2  Init array V to 0                    // semiring zero
    // One-dimensional preprocessing:
 3  for each A ∈ N_u   1D-Earley-Forward(A, u)
 4  for each A ∈ N_v   1D-Earley-Forward(A, v)
 5  for j := 1 to n + 1
 6      for l := 1 to m + 1
 7          for p := 1 to |R|           // iterate over rules
 8              2D-Predict-Forward(j, l, p)
 9          end for
10          for i := j downto 1
11              for k := l downto 1
12                  for p := 1 to |R|           // for all rules
13                      for q := 1 to ℓ(p)      // Start at 1: omit leftmost dot pos.
14                          2D-Complete-Forward((i j / k l), p, q)
15                      end for q
16                  end for p
17              end for k
18          end for i
19      end for l
20  end for j
```

Algorithm A.8 Handling of prediction items.

2D-PREDICT-FORWARD(j, l, p)

// Predict item $\left(\begin{smallmatrix} j & j \\ l & l \end{smallmatrix}, p, 0\right)$.

1 $jump := jumpLeft(p)$ // immediate scan
2 **if** $\ell(p) == 0$ // p is a terminating rule
3 $prob := P\left(p \wedge \begin{smallmatrix} u_{j,j+jump_1-1} \\ v_{l,l+jump_2-1} \end{smallmatrix}\right)$ // determine rule probability
// Set value including 4s for skipped terminals
4 $V[j + jump_1, l + jump_2, j, l, p, 0] := prob \cdot 4^{jump_1 + jump_2}$
5 **else**
6 $V[j + jump_1, l + jump_2, j, l, p, 0] := \mathbf{1} \cdot 4^{jump_1 + jump_2}$
7 **end if**

Algorithm A.9 Handling of completion items.

2D-Complete-Forward$\left(\begin{smallmatrix} i & j \\ k & l \end{smallmatrix}, p, q\right)$

// Asserts $1 \leqslant q \leqslant \ell(p)$, i. e. q may *not* be 0.
// Asserts that all needed item values are computed.
1 $B := \mathrm{nextNT}(p, q-1)$ // Get the non-terminal that was completed
2 **if** $q == \ell(p)$ // finalization
// We have to scan the terminals at right end of rule
3 $jump := jumpRight(p)$ // The offset for writing
4 $prob := \mathrm{P}\left(p \wedge \begin{smallmatrix} u_{i,j+jump_1-1} \\ v_{k,l+jump_2-1} \end{smallmatrix}\right)$ // rule probability
5 **else** // no finalization
6 $jump := \left(\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right)$
7 $prob := \mathbf{1}$
8 **end if**
// Handle one-dimensional non-terminals ...
9 **if** $\mathrm{D}_{\neq\varepsilon}(B) == \{1\}$
10 **for** $r := i$ **to** j
11 **for each** $p_B \in \mathrm{ind's}(B)$ // rules with left hand side B in G_B
12 $\mathtt{V}\left[j + jump_1, l + jump_2, i, k, p, q\right] \oplus=$
$\mathtt{V}\left[r, l, i, k, p, q-1\right] \otimes \mathtt{Vone}_B\left[j, r, p_B, \ell(p_B)\right]$
13 **end for** B-rule
14 **end for** r
15 **elseif** $\mathrm{D}_{\neq\varepsilon}(B) == \{2\}$
16 **for** $s := k$ **to** l
17 **for each** $p_B \in \mathrm{ind's}(B)$ // rules with left hand side B in G_B
18 $\mathtt{V}\left[j + jump_1, l + jump_2, i, k, p, q\right] \oplus=$
$\mathtt{V}\left[j, s, i, k, p, q-1\right] \otimes \mathtt{Vone}_B\left[l, s, p_B, \ell(p_B)\right]$
19 **end for** B-rule
20 **end for** r
// ... and two-dimensional non-terminals
21 **else** // $\rightsquigarrow \mathrm{D}_{\neq\varepsilon}(B) == \{1, 2\}$
22 **for** $r := i$ **to** j
23 **for** $s := k$ **to** l
24 **for each** $p_B \in \mathrm{ind's}(B)$ // rules with left hand side B in G
25 $\mathtt{V}\left[j + jump_1, l + jump_2, i, k, p, q\right] \oplus=$
$\mathtt{V}\left[r, s, i, k, p, q-1\right] \otimes \mathtt{V}\left[j, l, r, s, p_B, \ell(p_B)\right]$
26 **end for** B-rule
27 **end for** s
28 **end for** r
29 **end if**
// Value complete, include rule probability now
30 $\mathtt{V}\left[j + jump_1, l + jump_2, i, k, p, q\right] \otimes= prob \cdot 4^{jump_1 + jump_2}$

A.6. 2D-Earley-Reverse

Algorithm A.10 Main loop in reverse parse, over items in $\succ$ order.

2D-Earley-Backward(u, v)

```
    // S′ is the start symbol of the grammar G to use.
    // S′ → S the only rule expanding S.
    // u and v are the two components of the input pair.
    // Assumes V, Vone are filled with forward values.
 1  n := |u|   ,   m := |v|
 2  Init array Z to 0
 3  Z[n+1, m+1, 1, 1, ind(S′ → S), 1] := 1
 4  for j := n+1 downto 1
 5      for l := m+1 downto 1
 6          for i := 1 to j
 7              for k := 1 to l
 8                  for p := |R| downto 1
 9                      2D-Complete-Second-Type( i j / k l , p )
10                      for q := ℓ(p) − 1 downto 1      // ⇝ if ℓ(p) == 0, skip loop
11                          2D-Complete-First-Type( i j / k l , p, q )
12                      end for q
13                  end for p
14              end for k
15          end for i
16      end for l
17  end for j
```

Algorithm A.11 Handling of completion, second type.

2D-Complete-Second-Type$\left(\begin{smallmatrix} i & j \\ k & l \end{smallmatrix}, p_B\right)$
// Creates item $\left(\begin{smallmatrix} i & j \\ k & l \end{smallmatrix}, p_B, \ell(p_B)\right)$.
1 $B := \mathit{source}(p_B)$ // B is left side of rule with index p_B. By assumption, $B \in N_2$.
2 $\mathit{prob} := P\left(p_B \wedge \begin{smallmatrix} u_{i,j-1} \\ v_{k,l-1} \end{smallmatrix}\right)$ // Include the probability for rule p_B
3 **for** $r := 1$ **to** i
4 **for** $s := 1$ **to** k
5 **for each** $(p, q) \in \mathit{dottedRules}(B)$ // the $(q+1)$st non-terminal in rule p is B
// maybe we have to skip terminals at right end
6 **if** $q == \ell(p) - 1$
7 $\mathit{jump} := \mathit{jumpRight}(p)$
8 **else**
9 $\mathit{jump} := \left(\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right)$
10 **end if**
11 $c := 4^{\mathit{jump}_1 + \mathit{jump}_2}$
// Looking at items $I_1 = \left(\begin{smallmatrix} r & i \\ s & k \end{smallmatrix}, p, q\right)$ and $I_2 = \left(\begin{smallmatrix} r & j+\mathit{jump}_1 \\ s & l+\mathit{jump}_2 \end{smallmatrix}, p, q+1\right)$.
12 $Z[j, l, i, k, p_B, \ell(p_B)] \oplus =$
$c \cdot Z[j + \mathit{jump}_1, l + \mathit{jump}_2, r, s, p, q+1] \otimes V[i, k, r, s, p, q] \otimes \mathit{prob}$
13 **end for** dotted rule (p, q)
14 **end for** s
15 **end for** r

Algorithm A.12 Handling of completion, first type.

2D-Complete-First-Type$\left(\begin{smallmatrix} i & j \\ k & l \end{smallmatrix}, p, q\right)$

// Creates item $\left(\begin{smallmatrix} i & j \\ k & l \end{smallmatrix}, p, q\right)$.
// Assumes, $0 < q < \ell(p)$.
1 $B := \mathsf{nextNT}(p, q)$
2 **if** $q == \ell(p) - 1$ $\quad jump := jumpRight(p)$
3 **else** $\quad jump := \left(\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right)$
4 $c := 4^{jump_1 + jump_2}$
5 **if** $D_{\neq\varepsilon}(B) == \{1\}$
6 **for** $r := j$ **to** $n + 1$
// Fixing $I_2 = \left(\begin{smallmatrix} i & r+jump_1 \\ k & l+jump_2 \end{smallmatrix}, p, q+1\right)$.
7 **for each** $p_B \in \mathsf{ind's}(B)$ // rules with left hand side B in G_B
// Fixing $I_1 = \left(\begin{smallmatrix} j & r \end{smallmatrix}, p_B, \ell(p_B)\right)$.
8 $Z\big[j, l, i, k, p, q\big] \oplus=$
$c \cdot Z\big[r + jump_1, l + jump_2, i, k, p, q+1\big] \otimes \mathtt{Vone}_B\big[r, j, p_B, \ell(p_B)\big]$
9 **end for** p_B
10 **end for** r
11 **elseif** $D_{\neq\varepsilon}(B) == \{2\}$
12 **for** $s := l$ **to** $m + 1$
// Fixing $I_2 = \left(\begin{smallmatrix} i & j+jump_1 \\ k & s+jump_2 \end{smallmatrix}, p, q+1\right)$.
13 **for each** $p_B \in \mathsf{ind's}(B)$ // rules with left hand side B in G_B
// Fixing $I_1 = \left(\begin{smallmatrix} l & s \end{smallmatrix}, p_B, \ell(p_B)\right)$.
14 $Z\big[j, l, i, k, p, q\big] \oplus=$
$c \cdot Z\big[j + jump_1, s + jump_2, i, k, p, q+1\big] \otimes \mathtt{Vone}_B\big[s, l, p_B, \ell(p_B)\big]$
15 **end for** p_B
16 **end for** r
17 **else** // $\rightsquigarrow D_{\neq\varepsilon}(B) == \{1, 2\}$
18 **for** $r := j$ **to** $n + 1$
19 **for** $s := l$ **to** $m + 1$
// Fixing $I_2 = \left(\begin{smallmatrix} i & r+jump_1 \\ k & s+jump_2 \end{smallmatrix}, p, q+1\right)$.
20 **for each** $p_B \in \mathsf{ind's}(B)$ // rules with left hand side B in G
// Fixing $I_1 = \left(\begin{smallmatrix} j & r \\ l & s \end{smallmatrix}, p_B, \ell(p_B)\right)$.
21 $Z\big[j, l, i, k, p, q\big] \oplus=$
$c \cdot Z\big[r + jump_1, s + jump_2, i, k, p, q+1\big] \otimes V\big[r, s, j, l, p_B, \ell(p_B)\big]$
22 **end for** p_B
23 **end for** s
24 **end for** r
25 **end if**

Bibliography

[AA00] L. Argaman and S. Altuvia, **2000**. *fhlA repression by OxyS RNA: kissing complex formation at two sites results in a stable antisense-target RNA complex*. Journal of Molecular Biology, 300(5):1101 – 1112. ISSN 0022-2836. doi: DOI:10.1006/jmbi.2000.3942. URL http://www.sciencedirect.com/science/article/B6WK7-45F518G-9F/2/6f453b6ef1eb60f8fc416a1dbd8c04e8.
(Cited on page 76.)

[AH02] J. Aycock and R. N. Horspool, **2002**. *Practical earley parsing*. The Computer Journal, 45(6):620–630. URL http://www3.oup.co.uk/computer_journal/hdb/Volume_45/Issue_06/450620.sgm.abs.html.
(Cited on page 45.)

[AKN+06] C. Alkan, E. Karakoc, J. H. Nadeau, S. C. Sahinalp and K. Zhang, **2006**. *RNA-RNA interaction prediction and antisense RNA target search*. Journal of Computational Biology, 13(2):267–282.
(Cited on pages 9 and 23.)

[Alb02] D. M. Albro, **2002**. *An Earley-style parser for multiple context-free grammars*. URL http://www.humnet.ucla.edu/people/albro/papers.html.
(Cited on page 28.)

[AZC05] M. Andronescu, Z. Zhang and A. Condon, **2005**. *Secondary structure prediction of interacting RNA molecules*. Journal of molecular biology, 345(5):987–1001.
(Cited on pages 6, 9, 72, 73, 75, 76 and 77.)

[CG98] Z. Chi and S. Geman, **1998**. *Estimation of probabilistic context-free grammars*. Computational Linguistics, 24(2):299–305. ISSN 0891-2017.
(Cited on pages 18, 19 and 26.)

[DE04] R. D. Dowell and S. R. Eddy, **2004**. *Evaluation of several lightweight stochastic context-free grammars for RNA secondary structure prediction*. ISSN 1471-2105. URL http://www.biomedcentral.com/1471-2105/5/71.
(Cited on pages 36, 38 and 73.)

[DEKM98] R. Durbin, S. Eddy, A. Krogh and G. Mitchison, **1998**. *Biological sequence analysis — Probabilistic models of proteins and nucleic acids*. Cambridge University Press, Cambridge.
(Cited on pages 9, 19 and 52.)

[Ear70] J. Earley, **1970**. *An efficient context-free parsing algorithm*. Communications of the ACM (CACM), 13(2).
(Cited on page 12.)

[GJ08] D. Grune and C. J. H. Jacobs, **2008**. *Parsing Techniques — A Practical Guide*. Springer, 2nd edition. ISBN 038720248X. URL http://www.cs.vu.nl/~dick/PT2Ed.html.
(Cited on pages 12 and 45.)

[Goo98] J. Goodman, **1998**. *Parsing inside-out*. Ph.D. thesis. URL http://arxiv.org/abs/cmp-lg/9805007.
(Cited on pages 20, 45, 50, 52, 55, 67, 68 and 85.)

[Goo99] J. Goodman, **1999**. *Semiring parsing*. Computational Linguistics, 25(4):573–605.
(Cited on pages 20, 45, 50, 55 and 67.)

[GT04] T. Geissmann and D. Touati, **2004**. *Hfq, a new chaperoning role: binding to messenger RNA determines access for small RNA regulator*. The EMBO Journal, 23(2):396.
(Cited on page 76.)

[HQRS09] F. W. D. Huang, J. Qin, C. M. Reidys and P. F. Stadler, **2009**. *Partition function and base pairing probabilities for RNA-RNA interaction prediction*. Bioinformatics, 25(20):2646.
(Cited on pages 36 and 80.)

[HQRS10] F. W. D. Huang, J. Qin, C. M. Reidys and P. F. Stadler, **2010**. *Target prediction and a statistical sampling algorithm for RNA-RNA interaction*. Bioinformatics, 26(2):175–181. URL http://dx.doi.org/10.1093/bioinformatics/btp635.
(Cited on pages 23, 36, 37, 38 and 73.)

[HU79] J. E. Hopcroft and J. D. Ullman, **1979**. *Introduction to automata theory, languages, and computation*. Addison-Wesley Reading, MA.
(Cited on page 11.)

[KAS09] Y. Kato, T. Akutsu and H. Seki, **2009**. *A grammatical approach to RNA-RNA interaction prediction*. Pattern Recognition, 42(4):531–538.
(Cited on pages 6, 9, 71, 72, 73, 74 and 75.)

[Kat07] Y. Kato, **2007**. *Formal grammars for describing RNA pseudoknotted structure and their application to structure analysis*. Ph.D. thesis.
(Cited on page 24.)

[LN10] U. Laube and M. E. Nebel, **2010**. *Maximum likelihood analysis of algorithms and data structures*. Theoretical Computer Science, 411(1):188–212.
(Cited on page 18.)

[LP00] R. B. Lyngsø and C. N. S. Pedersen, **2000**. *RNA pseudoknot prediction in energy-based models*. Journal of Computational Biology, 7(3-4):409–427.
(Cited on page 17.)

[McC90] J. S. McCaskill, **1990**. *The equilibrium partition function and base pair binding probabilities for RNA secondary structure*. Biopolymers, 29(6-7):1105–1119.
(Cited on pages 40 and 80.)

[NS10] M. E. Nebel and A. Scheid, **2010**. *Prediction of RNA Secondary Structure by Statistical Sampling Using SCFG Models*. Submitted.
(Cited on pages 40, 41 and 80.)

[PSE+96] J. Paillart, E. Skripkin, B. Ehresmann, C. Ehresmann and R. Marquet, **1996**. *A loop-loop "kissing" complex is the essential part of the dimer linkage of genomic HIV-1 RNA*. Proceedings of the National Academy of Sciences of the United States of America, 93(11):5572.
(Cited on pages 72 and 77.)

[SBS09] R. Salari, R. Backofen and S. C. Sahinalp, **2009**. *Fast prediction of RNA-RNA interaction*. In S. Salzberg and T. Warnow, editors, *WABI*, volume 5724 of *Lecture Notes in Computer Science*, pages 261–272. Springer. ISBN 978-3-642-04240-9. URL http://dx.doi.org/10.1007/978-3-642-04241-6.
(Cited on page 9.)

[SMFK91] H. Seki, T. Matsumura, M. Fujii and T. Kasami, **1991**. *On multiple context-free grammars*. Theoretical Computer Science, 88(2):229. ISSN 0304-3975. doi: http://dx.doi.org/10.1016/0304-3975(91)90374-B.
(Cited on pages 24, 25 and 28.)

[UDV+05] K. Udekwu, F. Darfeuille, J. Vogel, J. Reimegård, E. Holmqvist and E. Wagner, **2005**. *Hfq-dependent regulation of OmpA synthesis is mediated by an antisense RNA*. Genes & development, 19(19):2355.
(Cited on pages 72 and 77.)

[WF02] E. G. H. Wagner and K. Flärdh, **2002**. *Antisense RNAs everywhere?* TRENDS in Genetics, 18(5):223–226.
(Cited on page 72.)

[WN10] F. Weinberg and M. E. Nebel, **2010**. *Extending stochastic context-free grammars for an application in bioinformatics*. In A. H. Dediu, H. Fernau and C. Martín-Vide, editors, *LATA*, volume 6031 of *Lecture Notes in Computer Science*, pages 585–595. Springer. ISBN 978-3-642-13088-5. URL http://dx.doi.org/10.1007/978-3-642-13089-2.
(Cited on page 80.)

[ZS84] M. Zuker and D. Sankoff, **1984**. *RNA secondary structures and their prediction*. Bulletin of Mathematical Biology, 46(4):591–621.
(Cited on page 23.)