Product Grammars for Alignment and Folding

Christian Höner zu Siederdissen, Ivo L. Hofacker, and Peter F. Stadler

Abstract—We develop a theory of algebraic operations over linear and context-free grammars that makes it possible to combine simple "atomic" grammars operating on single sequences into complex, multi-dimensional grammars. We demonstrate the utility of this framework by constructing the search spaces of complex alignment problems on multiple input sequences explicitly as algebraic expressions of very simple one-dimensional grammars. In particular, we provide a fully worked frameshift-aware, semiglobal DNA-protein alignment algorithm whose grammar is composed of products of small, atomic grammars. The compiler accompanying our theory makes it easy to experiment with the combination of multiple grammars and different operations. Composite grammars can be written out in $\LaTeX$ for documentation and as a guide to implementation of dynamic programming algorithms. An embedding in Haskell as a domain-specific language makes the theory directly accessible to writing and using grammar products without the detour of an external compiler. Software and supplemental files available here: http://www.bioinf.uni-leipzig.de/Software/gramprod/

Index Terms—linear grammar, context free grammar, product structure, multiple alignment, Haskell

1 Introduction

The well-known dynamic programming (DP) algorithms for the simultaneous alignment of n sequences [1] have a structure that is reminiscent of topological product structures. This is expressed e.g., by the fact that intermediary tables are n-dimensional. Here we explore whether this intuition can be made precise and operational. To this end we build on the conceptual framework of Algebraic Dynamic Programming (ADP) [2], [3]. In this setting a dynamic programming algorithm is separated into a context-free grammar (CFG) that generates the search space and an evaluation algebra. In this contribution we will mainly be concerned with a notion of product grammars to facilitate the construction of the search space.

Recent advances in RNA folding with pseudoknots [4], RNA-RNA interactions [5], [6], or RNA consensus structure prediction [7] have lead to the design of dynamic programming algorithms with dozens of intermediate tables. Their direct implementation in C or C++ is a major effort that is not only time-consuming but also error prone. The framework of algebraic dynamic programming could improve this situation considerably, but still does not provide a satisfactory solution because even the underlying grammars with nearly a hundred non-terminals are non-trivial to check. It is impossible to explore variants and refinements of these algorithms without major programming efforts unless ways and means can be found to construct the underlying grammars in a modular fashion. Product constructions, as introduced in [8] and significantly expanded in the present work, are one promising approach towards this end.

Before we delve into a more formal presentation, consider the context-free grammar for pairwise sequence alignment with affine gap costs as an example. Gotoh's algorithm [9] uses three non-terminals M, D, I, depending on whether the right end of the alignment is a match state, a gap in the first sequence, or a gap in the second sequence. The corresponding productions are of the form

$$\begin{aligned} M &\to M\binom{u}{v} \mid D\binom{u}{v} \mid I\binom{u}{v} \mid \binom{\$}{\$} \\ D &\to M\binom{u}{-} \mid D\binom{u}{.} \mid I\binom{u}{-} \\ I &\to M\binom{-}{v} \mid D\binom{-}{v} \mid I\binom{.}{v}, \end{aligned} \tag{1}$$

where u and v denote terminal symbols, '$-$' corresponds to gap opening, while '.' denotes the (differently scored) gap extension. The \$ here takes the role of the "sentinel character", i.e., matches the end of the input. Each of the non-terminals reads simultaneously from two separate input tapes. To make this property more transparent in the notation, we write $M \rightsquigarrow \binom{X}{X}$, $D \rightsquigarrow \binom{X}{Y}$, and $I \rightsquigarrow \binom{Y}{X}$. This yields productions such as

$$\begin{aligned} \binom{X}{X} &\to \binom{X}{X}\binom{u}{v} \simeq \binom{Xu}{Xv} \qquad \text{or} \\ \binom{Y}{X} &\to \binom{X}{Y}\binom{-}{v} \simeq \binom{X-}{Yv}. \end{aligned} \tag{2}$$

Apart from the conspicuous absence of $\binom{Y}{Y}$, i.e., alignments ending in an all-gap column, to which we will return later, this notation strongly suggests to consider the 1-dimensional projections of the two-dimensional productions of

- C. Höner zu Siederdissen is with the Department of Theoretical Chemistry University of Vienna, Währingerstrasse 17, A-1090, Vienna, Austria. E-mail: choener@tbi.univie.ac.at.
- I.L. Hofacker is with the Department of Theoretical Chemistry, University of Vienna, Währingerstrasse 17, A-1090, Vienna, Austria, the Research Group Bioinformatics and Computational, Währingerstrasse 29 A-1090, Vienna, and the Center for RNA in Technology and Health, University of Copenhagen, Grønnegå rdsvej 3, Frederiksberg C, Denmark. E-mail: ivo@tbi.univie.ac.at.
- P.F. Stadler is with the Bioinformatics Group of the Department of Computer Science and with the Interdisciplinary Center for Bioinformatics of the University of Leipzig, D-04107, Leipzig, Germany, the Max Planck Institute for Mathematics in the Sciences, Inselstraße 22, D-04103, Leipzig, Germany, the Fraunhofer Institute for Cell Therapy and Immunology, Perlickstrasse 1, D-04103, Leipzig, Germany, the Department of Theoretical Chemistry of the University of Vienna, Währingerstrasse 17, A-1090, Vienna, the Center for RNA in Technology and Health, Univ. Copenhagen, Grønnegå rdsvej 3, Frederiksberg C, Denmark, and the Santa Fe Institute, 1399 Hyde Park Road, Santa Fe NM 87501. E-mail: studla@bioinf.uni-leipzig.de.

Manuscript received 12 Dec. 2013; revised 29 Apr. 2014; accepted 10 May 2014. Date of publication 21 May 2014; date of current version 1 June 2015.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TCBB.2014.2326155

1545-5963 © 2014 IEEE. Translations and content mining are permitted for academic research only. Personal use is also permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

Eq. (2), which obviously have the form

$$X \to Xu \,\big|\, Yu \,\big|\, \$ \qquad \text{and} \qquad Y \to Y. \,\big|\, X - . \tag{3}$$

This simple grammar either reads a symbol (non-terminal X) or it ignores it (non-terminal Y, with '.' extending a gap, '$-$' opening a gap). Each copy of the "step grammar" (3) operates on its own input tape. This example suggests that dynamic programming algorithms for alignment problems in general have a product-like structure. Indeed, n-way alignments can be seen as an n-fold product of the simple step grammar with itself.

The aim of the present work is more general than alignments. We introduce products of grammars as a very general framework to facilitate a more effective design of dynamic programming algorithms. This requires that we clarify the precise meaning of a product of CFGs. Since alignment algorithms are naturally expressed as left-linear CFGs we first develop a theory for this special case and demonstrate in some detail how our framework can facilitate the construction of complex alignment algorithms. As a showcase application we consider mixed nucleotide/protein alignments with frameshifts. We then proceed to explore possibilities to generalize the product construction to context free grammars in general and show that normal forms can be employed to guide such constructions. We find that the Greibach normal form (GNF) admits an associative product that conserves the normal form but does not subsume the direct product of linear grammars.

2 Algebraic Operations on Linear Grammars

2.1 Notation

A CFG $\mathcal{G} = (N, T, P, S)$ consists of a finite set N of non-terminals, a finite set T of terminals so that $N \cap T = \emptyset$, a set P of productions $X \to \alpha$ where $X \in N$ and $\alpha \in (T \cup N)^*$, and a start symbol $S \in N$. (It will be convenient below to consider also grammars without a start symbol S). Furthermore, we need at least one special symbol \$ denoting the empty string, an "empty production" $\varnothing$ and the ε symbol denoting a "none"-symbol. This symbol emits nothing (like \$). As a parsing symbol, however, it succeeds always (in contrast to \$, which only succeeds on empty substrings).

Single and multiple tapes. Below we will make use of the term *tape*. A (single) tape is an input sequence. Grammars operating simultaneously on $k > 1$ input sequences are called multi-tape grammars of dimension k [10]. All terminal symbols of a k-tape grammar are k-dimensional, $t \in \prod_i (T_i \cup \epsilon)$, where T_i is the alphabet of the ith tape. Non-terminals are not necessarily tied to individual tapes. It will be convenient in many cases, however, to use "multi-dimensional" symbols for the non-terminals to emphasize their semantics.

The sentinel terminal. The use of \$ is arguably optional as the rule $X \to t$ with $t \in T$ also terminates a derivation and it is possible to append a sentinel character as the last character to each input string. While there is no *formal* requirement for \$, an explicit treatment of the terminating case can be advantageous in practical implementations as it does not require adding a sentinel symbol to the alphabet.

The none *symbol.* The "none" symbol (ε), on the other hand is a purely formal symbol that is used here for notational convenience. As a terminal, it denotes the empty string. In multi-tape algorithms such as Needleman-Wunsch it denotes the deletion in an in-del event (by *not* reading a character). More generally, it allows a more compact notation by recasting several rules into a single common format, as in the case of a non-terminal followed by terminal in left-linear grammars or the Greibach normal form, see Section 3.4 below. Whenever ε-only symbols appear in projections (defined below), they can be savely eliminated. It is possible to avoid ε, albeit at the expense of a somewhat more lengthy presentation.

In the next two sections we will consider in particular left-linear grammars, i.e., those for which all productions are of the form $A \to Bx$ with $A, B \in N$ and $x \in T$.

The example of Gotoh's algorithm in the introductory section motivates us to introduce algebraic operations on grammars in a more systematic way. As a running example, we will use one of the simplest alignment algorithms. The Needleman-Wunsch algorithm [11] aligns two sequences $x_{1\dots n}$ and $y_{1\dots m}$ so that the sum of matches and in/del scores is maximized. The basic recursion over the memoization table T reads

$$T_{ij} = \max \begin{cases} T_{i-1,j} + d \\ T_{i,j-1} + d \\ T_{i-1,j-1} + m(x_i, y_j) \\ 0 \text{ if } i = 0 \text{ and } j = 0. \end{cases} \tag{4}$$

In the recursive scheme, the base case is given by the alignment of two empty substrings "on the left", while the other cases extend the already aligned part of the strings to the left. This slightly unusual variant of the algorithm was chosen to be identical to the grammatical description that follows. The first two cases denote an in/del operation with cost d, while $m(\,.\,,\,.\,)$ scores the (mis)match x_i with y_j.

A two-tape grammar equivalent to the recursion in Equ. (4) is

$$\binom{X}{Y} \to \binom{X}{Y}\binom{a}{\varepsilon} \,\big|\, \binom{X}{Y}\binom{\varepsilon}{a} \,\big|\, \binom{X}{Y}\binom{a}{a} \,\big|\, \binom{\$}{\$}. \tag{5}$$

There are several differences between the formulation in Eq. (4) and Eq. (5). The recursive formulation working on the memoization table T does not store the alignment directly but rather the *score* of each partial, optimal alignment. The grammatical description, on the other hand, describes the *search space* of all possible alignments without any notion of scoring. In addition, recursive descriptions usually include explicit annotations for base cases, here the empty alignment. The production rule $\binom{X}{Y} \to \binom{\$}{\$}$ has this role in our example. In general, grammatical descriptions abstract away certain implementation details. Some of these will, however, become important when constructing more complex grammars from simpler ones, as we shall see below.

Our task will be to construct Eq. (5) from even simpler, "atomic" constituents. These grammars are

$$\mathcal{S} = (\{X\}, \{a\}, \{X \to Xa \,\big|\, X\}, X)\,, \tag{6}$$

$$\mathcal{N} = (\{X\}, \{\$\}, \{X \to \$\}, X)\,, \tag{7}$$

$$\mathcal{L} = (\{X\}, \{\}, \{X \to X\}, X)\,. \tag{8}$$

The rules in Eqs. (6–8) can be associated with single-tape rules for Eq. (5) by writing $X \to X$ as $X \to X\varepsilon$, which conforms to the intuition built up in the introduction and Eq. (1). The grammar $\mathcal{S}$ in Eq. (6) performs a "step". It either reads a single character on the right and recurses on the left, or simply recurses. $\mathcal{S}$ describes the action of an alignment algorithm as seen by a single tape, i.e., without knowledge to what the recursion is doing simultaneously on other tapes. Note that by itself these rules do not terminate. The grammar $\mathcal{N}$, Eq. (7), matches the empty input (or any empty substring of the input) and immediately terminates. Finally, $\mathcal{L}$ (Eq. (8)) singles out the non-terminating loop case already seen in Eq. (6). Intuitively, we can combine these three components on a single tape as

$$\mathcal{S} + \mathcal{N} - \mathcal{L} = (\{X\}, \{a, \$\}, \{X \to Xa \mid \$\}, X) \tag{9}$$

to obtain a grammar that simply reads an input tape. While not particularly useful in its own right, $\mathcal{S} + \mathcal{N} - \mathcal{L}$ is the single-tape analog of an alignment grammar. More importantly, it highlights the convenience of algebraic operations on grammars, which we shall introduce rigorously below for linear grammars.

Each operator introduced below primarily acts on sets of production rules. They implicitly carry over to the involved sets of terminals and non-terminals in an obvious manner. Two production rules are equivalent if they are isomorphic as in Eq. (14). This is of relevance insofar that it leads to idempotency in one of the operators below, but does not otherwise interfere with parsing.[1] In the following we use the notation P^n to emphasize that the productions operate on n tapes. We will refer to $\dim \mathcal{G} = n$ as the dimension of the grammar.

2.2 Algebraic Operations on Grammars

The + monoid. The + operator is defined as the union of all production rules of the two grammars P_1^n and P_2^n each of dimension n:

$$P_1^n + P_2^n = P_1^n \cup P_2^n. \tag{10}$$

We enforce explicitly that the + operator requires that the two operand grammars have the same dimensionality. The + operation forms a monoid over the set of production rules. Since the production rules form a set, isomorphic rules collapse to a single rule. The empty set $P^n = \{\}$ is a neutral element and $P^n + P^n = P^n$, i.e., the + monoid is idempotent. Isomorphism on production rules is also symbolic, that is, $X \to X$ is isomorphic to $X \to X$ but not to $\{X \to Y, Y \to X\}$, even though the latter set of two rules reduces to the first. For our example, we have $(X \to Xa \mid X) + (X \to \$) = (X \to Xa \mid X \mid \$)$.

Please note that the + operation has different semantics than the usually encountered union of two grammars. In particular, the union $g \cup h$ of two grammars is typically defined in such a way that the intersection of non-terminals is empty, i.e., each non-terminal is tagged with a unique identifier. This is in contrast to our definition of the + operation, where we explicitly treat symbols as equal if they have the same name.

The − operator. While the + operator unifies two sets of production rules, the − operator acts as a set difference operator

$$P_1^n - P_2^n = \{p \in P_1^n \,|\, p \notin P_2^n\}. \tag{11}$$

As for +, it requires operands of the same dimensionality. By construction, − is not associative. Thus does not form a semigroup but merely a magma. The empty set of production rules acts as the neutral element on the right. This operator is important to explicitly remove production rules that yield infinite derivations. In our example, we need to remove $\{X \to X\}$. With the help of − we can write $(X \to Xa \mid X) - (X \to X) = (X \to Xa)$. We shall see that it is often convenient to "temporarily" introduce productions that later on are excluded again from the final algorithm.

The ⊗ monoid. The definition of a direct product of left linear grammars lies at the heart of this contribution.

Definition 1. *Let $\mathcal{G}_1 = (N_1, T_1, P_1, S_1)$ and $\mathcal{G}_2 = (N_2, T_2, P_2, S_2)$ be left-linear CFGs, i.e., all productions are of the form $A \to Bx$ or $A \to y$. Their direct product $\mathcal{G}_1 \otimes \mathcal{G}_2$ is the grammar $\mathcal{G} = (N, T, P, S)$ with non-terminals $N = N_1 \times N_2 \cup N_1 \times \{\varepsilon\} \cup \{\varepsilon\} \times N_2$, terminals $T = T_1 \times T_2 \cup T_1 \times \{\varepsilon\} \cup \{\varepsilon\} \times T_2$, the start symbol of the product is $S = \binom{S_1}{S_2}$. The productions are of the forms*

$$\begin{aligned}
\tbinom{A_1}{A_2} &\to \tbinom{B_1}{B_2}\tbinom{x_1}{x_2} \,\Big|\, \tbinom{B_1}{\varepsilon}\tbinom{x_1}{y_2} \,\Big|\, \tbinom{\varepsilon}{B_2}\tbinom{y_1}{x_2} \,\Big|\, \tbinom{y_1}{y_2} \\
\tbinom{A_1}{\varepsilon} &\to \tbinom{B_1}{\varepsilon}\tbinom{x_1}{\varepsilon} \,\Big|\, \tbinom{y_1}{\varepsilon} \\
\tbinom{\varepsilon}{A_2} &\to \tbinom{\varepsilon}{B_2}\tbinom{\varepsilon}{x_2} \,\Big|\, \tbinom{\varepsilon}{y_2},
\end{aligned} \tag{12}$$

where $A_1 \to B_1 x_1$ and $A_1 \to y_1$, are productions in P_1 and $A_2 \to B_2 x_2$ and $A_2 \to y_2$ are productions in P_2, respectively.

By construction $\mathcal{G}$ is again a left-linear CFG that now operates on two bands. It will be convenient to abuse the notation and write productions of the form $A_i \to y_i$ as $A_i \to \varepsilon y_i$. Hence all productions in the product grammar can be written as $\binom{A_1}{A_2} \to \binom{B_1}{B_2}\binom{x_1}{x_2}$ with $A_i, B_i \in N_i \cup \{\varepsilon\}$, $x_i \in T_i \cup \{\varepsilon\}$ subject to the following conditions: $A_i = \varepsilon$ implies $B_i = x_i = \varepsilon$, $\binom{A_1}{A_2} \neq \binom{\varepsilon}{\varepsilon}$, and $\binom{\varepsilon}{\varepsilon}$ on the r.h.s. is omitted. We will also make use of notation $(A_1 \to B_1 y_1) \otimes (A_2 \to B_2 y_2)$ for the product of two individual productions. By construction, we have

$$\dim(\mathcal{G}_1 \otimes \mathcal{G}_2) = \dim\, \mathcal{G}_1 + \dim\, \mathcal{G}_2. \tag{13}$$

The empty string ε in the two-dimensional terminals and non-terminals is not necessarily associated with terminating the reading from the input band(s) as it denotes the absence of a parsing symbol. The \$ terminal symbol, on the other hand, explicitly parses only the empty (sub)-string.

To see that ⊗ is associative we need to demonstrate that the productions of $(\mathcal{G}_1 \otimes \mathcal{G}_2) \otimes \mathcal{G}_3$ and $\mathcal{G}_1 \otimes (\mathcal{G}_2 \otimes \mathcal{G}_3)$ are isomorphic, i.e.,

$$\left(\begin{matrix}\binom{x_1}{x_2}\\ x_3\end{matrix}\right) \to \left(\begin{matrix}\binom{\alpha_1}{\alpha_2}\\ \alpha_3\end{matrix}\right) \qquad \simeq \qquad \left(\begin{matrix}x_1\\ \binom{x_2}{x_3}\end{matrix}\right) \to \left(\begin{matrix}\alpha_1\\ \binom{\alpha_2}{\alpha_3}\end{matrix}\right). \tag{14}$$

[1] This is not completely true in the context of stochastic linear grammars: replication of a rule in an SCFG that already has duplicated rules requires that we sum over the probabilities for isomorphic rules.

This is most easily seen in the notation with the extra ε symbols since in this case the α_i are strings of length 2 that are simply decomposed in a column-wise fashion. Hence multiple products are well-defined. Furthermore, permutations of rows are isomorphisms. Thus $\mathcal{G}_1 \otimes \mathcal{G}_2 \simeq \mathcal{G}_2 \otimes \mathcal{G}_1$, i.e., exchanging the order of factors affects the order of the coordinates only. Due to the associativity of $\otimes$, we can safely extend these constructions to more than two factors. One easily checks that $\otimes$ and $+$ are distributive, i.e., $(\mathcal{G}_1 + \mathcal{G}'_1) \otimes (\mathcal{G}_2 + \mathcal{G}'_2) = \mathcal{G}_1 \otimes \mathcal{G}_2 + \mathcal{G}_1 \otimes \mathcal{G}'_2 + \mathcal{G}'_1 \otimes \mathcal{G}_2 + \mathcal{G}'_1 \otimes \mathcal{G}'_2$.

The canonical projection $\pi_i : \mathcal{G}_1 \otimes \mathcal{G}_2 \to \mathcal{G}_i$ is obtained by formally isolating the ith coordinate and contracting the empty strings ε and the empty productions $\varnothing = (\varepsilon \to \varepsilon)$. Clearly we have $\pi_i(T) = T_i$, $\pi_i(N) = N_i$, $\pi_i(S) = S_i$, and $\pi_i(P) = P_i$. The grammar product $\otimes$ thus has the basic properties of a well-defined product.

Let $lan(\mathcal{G})$ denote the language generated by G. Note that a "string" in $lan(\mathcal{G})$ is, by construction, a sequence of terminals, each of which is either of the form $\binom{x_1}{x_2}$ with $x_1 \in T_1$ and $x_2 \in T_2$, or of the form $\binom{x_1}{\varepsilon}$ with $x_1 \in T_1$, or of the form $\binom{\varepsilon}{x_2}$ with $x_2 \in T_2$. Thus $lan(\mathcal{G}_1 \otimes \mathcal{G}_2)$ consists of alignments of strings $\alpha_i \in \mathcal{G}_i$. To see this, note that each string $\alpha_i \in \mathcal{G}_i$ is generated from s_i using a finite sequence $\wp_i = (p_i^1, p_i^2, \ldots)$ of productions. Any partial matching of the $\wp_1$ and $\wp_2$ that preserves the sequential order of the two input sequences gives rise to a sequence $\wp$ of productions of the product grammar by matching all unmatched p_i^k with the empty production $\varnothing$. By construction $\pi_i(\wp) = \wp_i$, i.e., $\wp$ derives an alignment of the input strings β_1 and β_2. Conversely, given a sequence $\wp$ of productions of the product grammar, we know that $\pi_i(\wp)$ is a sequence of productions of $\mathcal{G}_i$; hence it constructs strings in $lan(\mathcal{G}_i)$. It follows that the product language satisfies

$$\pi_i(\mathrm{lan}(\mathcal{G}_1 \otimes \mathcal{G}_2)) = \mathrm{lan}(\mathcal{G}_i). \tag{15}$$

Similarly, we find that parse trees have a natural alignment structure. Let τ be a parse tree for an input $\beta \in \mathrm{lan}(\mathcal{G}_1 \otimes \mathcal{G}_2)$. Its interior nodes are labeled by the productions, i.e., pairs of the form $\binom{A_1 \to B_1 x_1}{A_2 \to B_2 x_2}$, $\binom{A_1 \to B_1 x_1}{\varepsilon}$, or $\binom{\varepsilon}{A_2 \to B_2 x_2}$. The projections $\pi_i(\tau)$ are explained by retaining only the ith coordinate of the vertex label and contracting all vertices labeled by ε in $\pi_i(\tau)$ yields a valid parse tree for $\pi_i(\beta)$ w.r.t. $\mathcal{G}_i$. Thus τ is a tree alignment of the parse trees for the two input strings.

The direct product $\otimes$ forms a monoid on grammars with arbitrary dimensions since

$$P_1^m \otimes P_2^n = \left\{ (p_1 \otimes p_2)^{m+n} \,|\, p_1^m \in P_1^m, p_2^n \in P_2^n \right\}, \tag{16}$$

where $p_1 \otimes p_2$ is explained in Definition 1. The neutral element of the $\otimes$ monoid is the zero-dimensional grammar which has one production rule $\varepsilon^0 \to \varepsilon^0$ that neither reads nor writes anything as it does not operate on a tape. Albeit rather artificial at first glance, it is useful to have a neutral element available. For our example, we have

$$\begin{aligned} &(X \to Xa \,|\, X) \otimes (X \to Xa \,|\, X) \\ &\quad = \tbinom{X}{X} \to \tbinom{X}{X}\tbinom{a}{a} \mid \tbinom{X}{X}\tbinom{a}{\varepsilon} \mid \tbinom{X}{X}\tbinom{\varepsilon}{a} \mid \tbinom{X}{X}. \end{aligned} \tag{17}$$

This grammar contains the two-dimensional loop rule $\binom{X}{X} \to \binom{X}{X}$, derived from $(X \to X) \otimes (X \to X)$ that eventually needs to be eliminated. To this end, it will be convenient to consider yet another operation on productions.

The structure-preserving power $*$ For any k-dimensional grammar $\mathcal{G}$ and any natural number $n \in \mathbb{Z}$, $\mathcal{G} * n$ denotes the $k \times n$-dimensional grammar with the same structure. Each k-dimensional (terminal or non-terminal) symbol $(\alpha_1, \ldots, \alpha_k)^\top$ is transformed to an $k \times n$-dimensional symbol $((\alpha_1 \ldots \alpha_k), \ldots, (\alpha_1 \ldots \alpha_k))^\top$. Note that for a grammar with a single production rule we have $G \otimes G \equiv G * 2$.

For our example grammar, this operation is useful as short-hand for both Eq. (7) and Eq. (8). In the case of linear grammars, the $*$ operator is mostly useful as shorthand to expand singleton grammars. It is worth noting, however, that some algorithms in computational biology, notably the Sankoff algorithm [12], work on multiple tapes with a grammar structured very similar to the one-dimensional relatives. We will return to the topic in Section 3.3.

2.3 The Needleman-Wunsch Alignment Grammar

We can now construct the full Needleman-Wunsch alignment grammar from the much simpler 1-dimensional constituents of Eqs.(6–8) in the following way:

$$\mathcal{NW} = \mathcal{S} \otimes \mathcal{S} + \mathcal{N} * 2 - \mathcal{L} * 2\,. \tag{18}$$

Written in terms of the productions only, this can be rephrased as

$$\begin{aligned} &(X \to Xa|X) \otimes (X \to Xa|X) \\ &\quad + (X \to \$) * 2 - (X \to X) * 2 \\ &= \tbinom{X}{X} \to \tbinom{X}{X}\tbinom{a}{a} \mid \tbinom{X}{X}\tbinom{a}{\varepsilon} \mid \tbinom{X}{X}\tbinom{\varepsilon}{a} \mid \tbinom{\$}{\$}. \end{aligned} \tag{19}$$

Again we have used a distinct symbol \$ to highlight the termination case deriving from $\mathcal{N}$. Since our construction of the Needleman-Wunsch grammar is based on well-defined algebraic operations we can readily use the same approach to construct much more complex alignment algorithms. Before we proceed, however, we provide with Gotohs algorithm a more complex example and address the technical issue of loop rules.

2.4 Gotoh's Grammar in Product Form

The construction of Gotoh's algorithm (Eq. 1) is a bit more complicated than the simple product construction for the Needleman-Wunsch grammar. It will make use of the following simple component grammars, of which grammar S (Eq. 24) defines a start symbol:

$$\mathcal{G} = (\{X, Y\}, \{a\}, \{X \to Xa \mid Ya, Y \to X \mid Y\})\,, \tag{20}$$

$$\mathcal{V} = (\{X, Y, S\}, \{a\}, \{X \to Ya, Y \to Y, S \to Y\})\,, \tag{21}$$

$$\mathcal{W} = (\{X, Y\}, \{\}, \{Y \to X \mid Y\})\,, \tag{22}$$

$$\mathcal{N} = (\{X\}, \{\$\}, \{X \to \$\})\,, \tag{23}$$

$$\mathcal{S} = (\{X, Y, S\}, \{\}, \{S \to X \mid Y\}, S)\,, \tag{24}$$

Grammar $\mathcal{G}$ encodes the generic "step" and distinguishes between "non-gap" state X and gap state Y. Correspondingly, the two-tape product $\mathcal{G} \otimes \mathcal{G}$ generates the core of the pairwise Gotoh grammar. The rules $X \to Xa \mid Ya$ align a character, while $Y \to X \mid Y$ denotes a deletion. Gap opening or gap extension costs are assigned to $Y \to X$ and $Y \to Y$, respectively. $\mathcal{G} \otimes \mathcal{G}$ has the 16 two-tape productions

$$\begin{aligned}
\binom{X}{X} &\to \binom{X}{X}\binom{a}{a} \mid \binom{X}{Y}\binom{a}{a} \mid \binom{Y}{X}\binom{a}{a} \mid \binom{Y}{Y}\binom{a}{a}\\
\binom{X}{Y} &\to \binom{X}{X}\binom{a}{\varepsilon} \mid \binom{X}{Y}\binom{a}{\varepsilon} \mid \binom{Y}{X}\binom{a}{\varepsilon} \mid \binom{Y}{Y}\binom{a}{\varepsilon}\\
\binom{Y}{X} &\to \binom{X}{X}\binom{\varepsilon}{a} \mid \binom{X}{Y}\binom{\varepsilon}{a} \mid \binom{Y}{X}\binom{\varepsilon}{a} \mid \binom{Y}{Y}\binom{\varepsilon}{a}\\
\binom{Y}{Y} &\to \binom{X}{X} \qquad \mid \binom{X}{Y} \qquad \mid \binom{Y}{X} \qquad \mid \binom{Y}{Y}\,.
\end{aligned} \tag{25}$$

There are seven productions that involve the semantically meaningless non-terminal $\binom{Y}{Y}$, which would denote a deletion on *both* tapes. $\mathcal{S} \otimes \mathcal{S}$ introduces transitions from the start symbol. The grammar $\mathcal{V} \otimes \mathcal{V}$ comprises exactly the rules that produce $\binom{Y}{Y}$, while $\mathcal{W} \otimes \mathcal{W}$ provides the production with $\binom{Y}{Y}$ on their left hand side. Subtracting these from $\mathcal{G} \otimes \mathcal{G}$ leaves the well known transitions of Gotoh's algorithm. It only remains to add the termination rule $\mathcal{N} * 2$. This construction immediately generalized to arbitrary dimensions: the k-tape Gotoh grammar can be written as

$$\bigotimes_1^k \mathcal{G} + \bigotimes_1^k \mathcal{S} - \bigotimes_1^k \mathcal{V} - \bigotimes_1^k \mathcal{W} + \mathcal{N} * k\,. \tag{26}$$

In this notation, alignment grammars for affine gap cost functions can be generated easily for any fixed dimension k. Already for $k = 3$ this task is quite a bit of chore if it has to be done manually.

Specialized to $k = 2$, Eq. (26) evaluates to

$$\begin{aligned}
\binom{S}{S} &\to \binom{X}{X} \qquad \mid \binom{X}{Y} \qquad \mid \binom{Y}{X}\\
\binom{X}{X} &\to \binom{X}{X}\binom{a}{a} \mid \binom{X}{Y}\binom{a}{a} \mid \binom{Y}{X}\binom{a}{a} \mid \binom{\$}{\$}\\
\binom{X}{Y} &\to \binom{X}{X}\binom{a}{\varepsilon} \mid \binom{X}{Y}\binom{a}{\varepsilon} \mid \binom{Y}{X}\binom{a}{\varepsilon}\\
\binom{Y}{X} &\to \binom{X}{X}\binom{\varepsilon}{a} \mid \binom{X}{Y}\binom{\varepsilon}{a} \mid \binom{Y}{X}\binom{\varepsilon}{a}.
\end{aligned} \tag{27}$$

Here, $\binom{X}{X}$ corresponds to the match M in Eq. (1), while D is replaced by $\binom{X}{Y}$ and I by $\binom{Y}{X}$. This notation has the additional advantage that it makes explicit on which tape(s) a match or insertion takes place, since for $k \geq 3$ the three non-terminals for match (M), deletion (D), insertion (I) are not sufficient. We also include an explicit start symbol $\binom{S}{S}$ now. The distinction between a gap opening and a gap extension is encoded at the level of the tagged rules. Compared to Eq. (1) we do not distinguish terminal symbols '.' and '–' anymore. Instead we use ε to denote a missing character without incurring a loss of expressability.

It could be argued that both V and W are unnecessary if the production rules are somehow tagged as "illegal" in the scoring algebra. We suggest, however, to always remove unwanted production rules (and non-terminals) explicitly. This ensures that the grammar describes exactly the search space under consideration in a formally correct manner and reduces the danger of introducing bugs into the implementation by, say, accidentally evaluating an "illegal" rule with a finite score.

2.5 Grammars with Loops

In Eq. (18) we explicitly added a terminating base case $X \to \$$ and removed a production rule with infinite derivations $X \to X$, or, equivalently $X \to X\varepsilon$. Why do we insist on performing this operation explicitly instead of modifying the definition of the direct product $\otimes$ accordingly?

The main reason lies in performance considerations. An "intelligent" product operator would first need to determine which rules have infinite derivations. For linear grammars with only one non-terminal a rule is not infinite if a single terminal (except ε) is present. \$ rules are also fine, as long as only the empty word case $X \to \$$ is present. Productions of the form $\{X \to Y, Y \to X\}$, however need to be followed up to a depth of the number of production rules present. For context-free grammars, the complexity will increase further, as in general multiple non-terminals may exist on the right-hand side. For both convenience and efficiency (by a constant factor), it does not seem to be desirable to transform the grammar into Chomsky normal form (CNF). The second problem is the need for rewriting. In the case of $\{X \to Y,\, Y \to X\}$, rewriting yields $X \to X$ by inserting the rules for Y wherever Y is used. More complicated grammars might quite easily require major rewrites before all loop cases can be removed.

Finally, using looping productions can be conceptually useful during construction. In case of Eq. (6), we either want to read a character in a "step" $X \to Xa$ or perform an in/del with a "stand" $X \to X$. The direct product of Eq. (6) then yields all possibilities of stepping or standing on two (or more) tapes. Of these cases we only want to remove the case where all tapes "stand". This case is quite easily determined as Eq. (8) and just needs to be scaled (with $*$) to the correct dimension and subtracted from the complete grammar.

2.6 Implementation

We have implemented a small compiler for our grammar product formalism with four output targets. First, we generate $\LaTeX$ output. This supports researchers in the development of complex, multiple dimensional linear and context-free (in 2-GNF) grammars, facilitates the comparison with the intended model for an elaborate alignment-like algorithm. It assists implementation of the grammar in the users' programming language of choice as the mathematical description of the recurrences reduces the chance that a production rule or recursion is simply forgotten.

In addition, we directly target the functional programming language Haskell [13]. It is possible to emit a Haskell module prototype which then needs to be extended with user-defined evaluation (scoring) algebras. This mode mirrors the $\LaTeX$ output. Advanced users may make use of TemplateHaskell [14] and QuasiQuotation [15] to *directly embed* our domain-specific language as a proper extension of Haskell itself. Both Haskell-based approaches ultimately make use of stream fusion optimizations [16] by way of the `ADPfusion` [17] framework that produces efficient code for dynamic programming algorithms.

Currently, the emitted Haskell code for non-trivial applications is slower than optimized `C` by a factor of 2 [17].

Recent additions to the compiler infrastructure [18], which provide instruction-level parallelism, will reduce this factor further. As `ADPfusion` is built on top of the `Repa` [19] library for CPU-level parallelism, we can expect improvements in this regard to be available for our dynamic programming algorithms in the near future.

Finally, we provide colored, pretty-printed diagnostics to aid during grammar development.

3 Alignment Algorithms

The overwhelming majority of alignment programs solve pairwise alignment problems by exact DP but use heuristics to combine the pairwise solutions to multiple alignments. The main reason is practicality. Full-fledged n-way DP alignments have exponential running time in n and hence are of little practical use for large n despite of elaborate divide-and-conquer strategies have been proposed to prune the search space, see e.g., [1]. Three-way alignments nevertheless are employed in practise in particular when high accuracy is crucial, see e.g., [20], [21], [22], [23]. Four-way alignments were recently explored for aligning short words from human language data [24]. We suspect that DP approaches for moderate values of n have not been explored for specialized application because of the effort for their implementation. In this section we demonstrate how the product construction can help, using a combined nucleic-acid/protein alignment algorithm as an example.

3.1 Global, Semi-Global, and Local Alignments

3.1.1 Global Alignments

The global alignment described above is the simplest variant of pairwise sequence alignment. Needleman-Wunsch style global alignments in grammatical form have a very convenient structure. The global alignment of k sequences (and therefore k tapes) can be written as $\mathcal{SW}_k = \otimes_{i=1}^{k}\mathcal{S} - \mathcal{L} * k + \mathcal{N} * k$, where $\otimes_{i=1}^{k}\mathcal{S}$ denotes the k-fold product of a grammar with itself. By virtue of having a monoidal (and hence associative) structure of the $\otimes$ operator it is well-defined.

This property of the global alignment grammar was quite useful in recent work on historical linguistics [25] where all alignments for k-tuples with $k \in \{2,3,4\}$, or two- to four-way alignments, were required. Scoring in these grammars was done by algebras using the sum-of-pairs scheme. We will come back to these kinds of scoring schemes in the conclusion, as they open up ways to describe automatic generation of algebras[2] for grammar products.

In many applications one is interested in local alignments that allow prefixes and suffixes to remain unaligned. It is possible to perform a local alignment with an adapted scoring scheme (as done in the Smith-Waterman algorithm [26]). Within the grammar-centered framework explored here, however, it seems preferable to devise a grammar that describes such a local alignment. Below we consider two natural extensions that are practical importance as semi-global (glocal) or local alignment algorithms.

3.1.2 Semi-Global Alignments

We first modify the Needleman-Wunsch grammar, equ. (18), in such a way that it models a semi-global alignment, i.e., to allow the grammar to act locally on one or more tapes. This allows us to construct scanning-type algorithms that can be used in genome-wide applications such as `HMMer` [27] and `Infernal` [28], [29]. The basic idea is to replace the one-dimensional "step grammar" by a slightly more complex one that allows us to skip a prefix or a suffix:

$$\mathcal{S} = (\{X\}, \{a\}, \{X \to Xa \mid X\}) \tag{28}$$

$$\mathcal{N}_{\mathcal{L}} = (\{L\}, \{\$\}, \{L \to \$\}) \tag{29}$$

$$\mathcal{N}_{\mathcal{X}} = (\{X\}, \{\$\}, \{X \to \$\}) \tag{30}$$

$$\mathcal{L} = (\{X\}, \{\}, \{X \to X\}, X) \tag{31}$$

$$\begin{aligned}\mathcal{O} =& (\{X, R, L\}, \{a\},\\ &\{R \to Ra \mid X, X \to L, L \to La\}, R).\end{aligned} \tag{32}$$

The extension from a global to a semi-global (or global) alignment is done using another grammar with a total of four rules. These rules allow the removal of nucleotides on the right (via R rules) or left (L rules) and switching to and from the actual alignment grammar. The extended step grammar $\mathcal{O}$ introduces the transitions from the right "ignored" part R to the aligned part X, and finally from X to the left "ignored" part L. It reads through the "ignored" parts with $R \to Ra$ and $L \to La$ rules. The "stop grammar" $\mathcal{N}$ now needs to recognize the end of the tape \$ as the r.h.s. of the non-terminal L.

The combined semi-global grammar can be written as

$$\mathcal{SG} = \mathcal{S} \otimes \mathcal{S} + \mathcal{O} \otimes \mathcal{L} - \mathcal{L} * 2 + \mathcal{N}_{\mathcal{L}} \otimes \mathcal{N}_{\mathcal{X}} \tag{33}$$

has the eight productions

$$\begin{aligned}&\binom{R}{X} \to \binom{R}{X}\binom{a}{\varepsilon} \mid \binom{X}{X}\\ &\binom{X}{X} \to \binom{X}{X}\binom{a}{a} \mid \binom{X}{X}\binom{\varepsilon}{a} \mid \binom{X}{X}\binom{a}{\varepsilon} \mid \binom{L}{X}\\ &\binom{L}{X} \to \binom{L}{X}\binom{a}{\varepsilon} \mid \begin{pmatrix}\$\\ \$\end{pmatrix}\end{aligned} \tag{34}$$

with $\binom{R}{X}$ as the start symbol. It embeds the core of the alignment grammar, $\mathcal{S} \otimes \mathcal{S}$ into a head and tail part that steps through the first band only. By construction $\mathcal{SG}$ is local only on the first tape, and global on the second tape. Intuitively, we can understand it as $\mathcal{SG} \sim \mathcal{NW} + \mathcal{O} \otimes \mathcal{L}$, i.e., as the Needleman-Wunsch grammar plus the skipping of a prefix and suffix on the first tape.

3.1.3 Local Alignments

The Smith-Waterman algorithm [26] for local sequence alignment is usually implemented via the scoring scheme. Including a neutral element (i.e., 0 for max-optimizations where sub-scores are summed up) into the optimization function yields a local alignment algorithm. As for the semi-global alignment, we again employ a

[2] We do not use the term *algebra product* in this case as algebra products already describe well-defined combinations of algebras in ADP [2].

grammar-based scheme to derive a local algorithm from a global one. Our construction is based on the observation that we can interpret the Smith-Waterman algorithm as a "concatenation" of three interconnected Needleman-Wunsch algorithms, where the first and the last one score only the excluded parts of the sequences. This can be written as

$$\mathcal{SW} = +_{i=0}^{2} \left(\mathcal{S}_i \otimes \mathcal{S}_i - \mathcal{L}_i * 2\right) + \mathcal{N}_2 * 2 + \mathcal{T} * 2\,. \tag{35}$$

Here, $\mathcal{S}, \mathcal{L}, \mathcal{N}$ are derived from the global alignment versions

$$\begin{aligned}
\mathcal{S}_i &= (\{X_i\}, \{a\}, \{X_i \to X_i a \mid X_i\}) \\
\mathcal{L}_i &= (\{X_i\}, \{\}, \{X_i \to X_i\}) \\
\mathcal{N}_2 &= (\{X_2\}, \{\$\}, \{X_2 \to \$\}) \\
\mathcal{T} &= (\{X_0, X_1, X_2\}, \{\}, \{X_0 \to X_1, X_1 \to X_2\}, X_0)
\end{aligned}$$

and $\mathcal{T}$ defines the transitions between the grammars. The resulting product grammar contains 12 production rules with $\binom{X_0}{X_0}$ as start symbol:

$$\begin{aligned}
\begin{pmatrix} X_0 \\ X_0 \end{pmatrix} &\to \begin{pmatrix} X_1 \\ X_1 \end{pmatrix} \Big| \begin{pmatrix} X_0 \\ X_0 \end{pmatrix}\begin{pmatrix} \varepsilon \\ a \end{pmatrix} \Big| \begin{pmatrix} X_0 \\ X_0 \end{pmatrix}\begin{pmatrix} a \\ \varepsilon \end{pmatrix} \Big| \begin{pmatrix} X_0 \\ X_0 \end{pmatrix}\begin{pmatrix} a \\ a \end{pmatrix} \\
\begin{pmatrix} X_1 \\ X_1 \end{pmatrix} &\to \begin{pmatrix} X_2 \\ X_2 \end{pmatrix} \Big| \begin{pmatrix} X_1 \\ X_1 \end{pmatrix}\begin{pmatrix} \varepsilon \\ a \end{pmatrix} \Big| \begin{pmatrix} X_1 \\ X_1 \end{pmatrix}\begin{pmatrix} a \\ \varepsilon \end{pmatrix} \Big| \begin{pmatrix} X_1 \\ X_1 \end{pmatrix}\begin{pmatrix} a \\ a \end{pmatrix} \\
\begin{pmatrix} X_2 \\ X_2 \end{pmatrix} &\to \begin{pmatrix} \$ \\ \$ \end{pmatrix} \Big| \begin{pmatrix} X_2 \\ X_2 \end{pmatrix}\begin{pmatrix} \varepsilon \\ a \end{pmatrix} \Big| \begin{pmatrix} X_2 \\ X_2 \end{pmatrix}\begin{pmatrix} a \\ \varepsilon \end{pmatrix} \Big| \begin{pmatrix} X_2 \\ X_2 \end{pmatrix}\begin{pmatrix} a \\ a \end{pmatrix}.
\end{aligned} \tag{36}$$

The naïve formulation is not ideal in practise in that it requires three memoization tables for the three non-terminals $\binom{X_0}{X_0}$, $\binom{X_1}{X_1}$, and $\binom{X_2}{X_2}$. In case of a local scoring system where the excluded parts of the alignment ($\binom{X_0}{X_0}$ and $\binom{X_2}{X_2}$) are scored by a constant, it is possible to replace the $O(nm)$ memo-tables with tables of size $O(1)$. This is possible by recognizing that *every* subword (index) in such a table can be memoized by the same single value. We will come back to this point in the discussion section.

3.1.4 Symmetric and Asymmetric Scoring

It is important to recognize that the grammar alone is a device that enumerates *all* possible alignments of a DNA sequence with a protein sequence. In particular, the grammar itself will not disallow alignments that are biologically unsound. However, each grammar created using our framework has all of its rules tagged with function symbols. These function symbols are also known as algebra symbols in the context of ADP [2] where we also borrowed the tagging symbol '<<<' used in Fig. 1. In this sense, our framework is very similar to S-attribute grammars [30].

Nevertheless, we can support the construction of the scoring algebra already during grammar design be explicitly making use of symmetries in the model. The alignment of two sequences of the same type is usually simplified due to mirrored operations. Recalling the alignment grammar from above, we speak of in/del operations as an insertion in one sequence that may just as well be described as a deletion in the other sequence. In addition, it does not matter which sequence is bound to which input tape. In some applications this symmetry is broken. For example, ancient DNA is partially chemically degraded by cytosin deamination, i.e., C is misread as T in sequencing [31]; to model such effects, asymmetric substitution score matrices are required. The same is true for alignments of lexical data in a computational linguistics setting because sound changes are directional between two languages [24]. To enforce symmetry we may use the same non-terminal symbols for each tape, while asymmetry can be indicated by the use of different (or indexed) symbols on the different tapes.

3.2 DNA-Protein Alignment

The problem of aligning a protein sequence to a nucleic acid (RNA or DNA) sequence is a rather specialized problem that arises in particular in the context of (homology-based) gene annotation. The best example is probably NCBI's `prosplign`, which aligns a protein query sequence to a piece of genomic DNA allowing also for introns. A detailed description of this dynamic programming algorithm has not yet become available. An interesting variation on this theme is gene annotation in the presence of extensive insertion/deletion editing as observed in the mitochondria of *Physarum polycephalum* or trypanosomatids. Frequent changes of the reading frame make it virtually impossible to identify homologs of mitochondrial proteins by `tblastn`, thus calling for specialized alignment algorithms [32], [33].

Our task is to align the amino acid sequence of a protein that may be present in a mitochondrial genome to the entire nucleic acid sequence of a mitogenome. Since we suspect that mRNAs may be subject to insertion or deletion editing, it is necessary to track frameshifts. Fig. 1 shows a general version of such an approach. The DNA sequence is read in one of three reading frames (RFs), and a deletion or insertion does not yield a "simple" in/del but also a frame shift to account for the effect of in/dels on the translation of the DNA into protein according to the "codons" of the genetic code. In Fig. 1 frame shifts (with scoring functions `rf1` and `rf2`) are enabled. Staying within a frame is modelled either by a (mis)match `stay` or by the deletion of all three characters of a codon (`del`). Finally, the alignment is to be calculated locally w.r.t. the DNA sequence but globally w.r.t. the amino acid sequence. In the grammar of Fig. 1 this is achieved by adding a component grammar that "skips" an unaligned prefix and suffix on the DNA band while leaving the protein band untouched. This follows the same insight as in the simpler alignment grammars above.

As each of the three frames, and shifts to the other two frames, is by itself similar to the other two frames, a special encoding saves a lot of work. The F non-terminal indicating the current frame is indexed with indices 0, 1, and 2. Frame shifts are thus calculated *modulo 3* instead of explicitly creating all three frame indices F_0 to F_2 and their corresponding production rules. Furthermore, all alignments are local with respect to the DNA sequence but global with respect to the protein sequence. The product of an embedding grammar (`DNAlocal`) with a grammar that does not read any amino acid character yields the correct semi-global embedding. For the protein sequence, the corresponding `PROstand` grammar can simply be reused.

The complexity of the DNA-protein alignment stems from the fact that we need to "align" the different frame shifting possibilities in the DNA input while matching

```
Grammar: DNAlocal
N: F{3} R L
T: c e
S: R
R    -> skp <<< R c
R    -> lcl <<< F{i}
F{i} -> lcl <<< L
L    -> skp <<< L c
//

Grammar: DNA
N: F{3}
T: c
F{i} -> stay <<< F{i}   c c c
F{i} ->  rf1 <<< F{i+1} c c
F{i} ->  rf2 <<< F{i+2} c
F{i} ->  del <<< F{i}
//

Grammar: DNAdone
N: F{3} R L
T: e
F{i} -> nil <<< e
R    -> nil <<< e
L    -> nil <<< e
//

Grammar: DNAstand
N: F{3}
F{i} -> del <<< F{i}
//
```

```
Grammar: PRO
N: P
T: a e
P -> amino <<< P a
P -> del   <<< P
//

Grammar: PROdone
N: P
T: e
S: P
P -> nil <<< e
//

Grammar: PROstand
N: P
P -> del <<< P
//
```

```
Product: DnaPro
  DNA >< PRO
+ DNAlocal >< PROstand
+ DNAdone >< PROdone
- DNAstand >< PROstand
//
```

Fig. 1. Atomic grammars for the DNA-Protein alignment example. **(I)** Nucleotides are read in triplets (three nucleotides each). The genome is aligned locally to the complete amino acid sequence. Using `DNAlocal`, nucleotides can be removed from the left or right end of the DNA sequence. The choice between local or global alignment for each tape is made based on adding the grammar product `DNAlocal >< PROstand`. The `DNA` grammar switches between reading frames. `DNAdone` and `DNAstand` handle the terminating and looping case. **(II)** The `PRO`tein grammar works similarly, but reads only a single amino acid at a time. The expansion of the `DNA` grammar is more complicated, as the indexed non-terminal symbol F expands to three different non-terminals corresponding to the three possible reading frames. **(III)** The grammar product of `DNA` and `PRO`tein without the looping case "`stand`" and with the terminating case "`done`". In code, $><$ represents the direct product ($\otimes$). The resulting 34-production rule grammar is shown in the Supplemental Material, available online, together with an extended description.

zero to three nucleotides to zero or one amino acid in the protein input. In addition, once a frame shift has occurred all following alignments of three nucleotides against one amino acid are scored in the new reading frame until another frame shift occurs or the alignment is completed. Under normal circumstances, the scoring algebra for the DNA-Protein grammar will assign very high costs to frameshift productions `rf1` and `rf2`. In order to model the frequent cytosine insertions in *P. polycephalum*, however, we simply use a moderate or low penalty for `rf1` when the incomplete codon is corrected by the inclusion of a 'C' that is not encoded in the DNA.

Our framework simplifies the complexity of designing this algorithm considerably. While the *combined* grammar is highly complex, the individual grammars are rather simple. As already mentioned, the protein "stepping grammar" is one of the simplest possible ones. The DNA grammar is more complex as we need to handle stepping and frame shifts in all three reading frames. But considering that we allow indexed non-terminals and calculations on these indices (modulo 3 in the frame shift case), even the frame shift grammar has only four rules, just twice as much as the simplest stepping grammar.

The resulting 34-production rule grammar is easily calculated in our frame work. We emphasize that one may readily extend this grammar to allow for, say, an alignment of two DNA sequences with two protein sequences. This grammar can be *calculated* at basically no additional cost but would pose a daunting task if implemented by hand.

In addition to the 34-production rule grammar, a set of $(10+1)$ *function types*[3] is created. Each function type is associated with a pair of function symbols, one for the DNA and one for the protein tape; apart from the choice function (the "+1"). This signature, as it is called in Algebraic Dynamic Programming [2], [17], associates one type with one or more of the production rules. For example, the production rule created from the product of `F{i} -> stay <<< F {i} c c c` and `P -> amino <<< P a` creates, among others, the production rule $\binom{F_0}{P} \rightarrow \binom{F_0}{P}\binom{c}{a}\binom{c}{\varepsilon}\binom{c}{\varepsilon}$.

The corresponding evaluation function `stay_amino` has type $\mathbb{S} \times (\mathcal{N} \times \mathcal{A}) \times (\mathcal{N} \times \mathcal{E}) \times (\mathcal{N} \times \mathcal{E}) \rightarrow \mathbb{S}$. This type specifies that `stay_amino` accepts the score (of type $\mathbb{S}$) of the alignments as calculated up to this position followed by tuples of characters read from each tape. The first nucleotide character (of the set $\mathcal{N} = \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}\}$) is aligned with the corresponding amino acid character (of the set $\mathcal{A}$ of the 20 amino acids). The remaining two nucleotides are aligned with elements from the empty set ($\mathcal{E}$ which emits the "non-informative character"

[3] The 11th function type designates the optimizing choice function, which is part of every evaluation algebra.

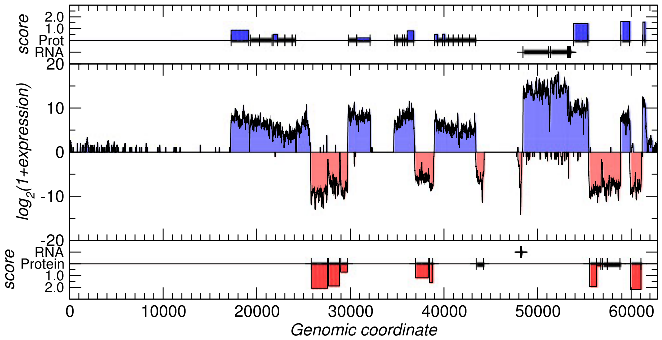

Fig. 2. Alignment of *R. americana* proteins to the *P. polycephalum* mitogenome. The central panel displays expression data from [35]. Above and below the known protein-coding (P) and ncRNA (R) genes are shown (thick black lines with delimiters for each gene) together with the alignment scores (normalized per nucleotide) for the *R. americana* proteins.

represented by the empty tuple `()`). This design allows us to effectively align the single amino acid character with three nucleotide characters. Finally, the `stay_amino` function emits a score of type $\mathbb{S}$.

General scoring for Frameshift-aware alignments: The user can now *implement* the required scoring functions. The scoring we describe here makes no assumptions on the knowledge of frequent cytosin (`C`) insertions. Instead we implement scoring for a *generic* frameshift-aware alignment algorithm which we then evaluate.

To this end we use three score lookup tables: (i) The alignment of three nucleotides to a single amino acid is performed by translating the codon into its respective amino acid, after which the, say, `BLOSUM` similarity matrices can be used to score the alignment. (ii) In case of a frame shift combined with a (mis)match either only one or two nucleotides are aligned with an amino acid. For, say, two nucleotides on the DNA sequence there are 12 possible codons: $\alpha c_1 c_2$, $c_1 \alpha c_2$, and $c_1 c_2 \alpha$, where $\alpha \in \{A, C, G, T\}$ and c_1, c_2 are the data of the DNA sequence. The insertion that maximizes the `BLOSUM` similarity is used to score the (mis) match. (iii) In addition to the `BLOSUM`-based scores, six gap parameters are required. Complete deletions of either a three-nucleotide codon or an amino acid ($g_{ccc} = -15$, $g_a = -10$), as well as the frame shift versions are penalized. To model the abundance of insertion editing sites, one-nucleotide frameshifts receive only a moderate penalty (approximately four strong mismatches (-20), partially offset by the `BLOSUM`-based match score for the repaired codon). Aligning a single nucleotide to an amino acid incurs a malus of -60, while nucleotide deletions incurring a frameshift are heavily penalized with a malus of -45 or -75. Finally, given that we want to align the protein semi-globally to the DNA sequence, transitions to and from the flanking part of the DNA have zero cost.

It is important to keep in mind that the generation of candidate alignments by the grammar is completely separate from the concrete scoring of the alignment by a scoring algebra. The amalgamation of the two concepts *grammar* and *scoring algebra* is taken care of by the `ADPfusion` framework [17], which also optimizes the resulting code such that its running time performance is competitive with handwritten `C`-based implementations.

3.2.1 Application of the Frame-Shift Grammar

To evaluate our DNA-Protein alignment algorithm we use a scenario as a test case in which frameshifts are rather frequent events. The mitochondrial genome of the amoeba *Physarum polycephalum* has long resisted comprehensive annotation because insertion editing is so frequent in most of its transcripts that `blastp`-based searches for homologs of known mitochondrial proteins have long remained unsuccessful [32]. This situation has changed only with the construction of a dedicated DNA/protein alignment algorithm that specifically modelled the C insertion [33], [34]. With the recent characterization of the mitochondrial transcriptome of *P. polycephalum* [35] a comprehensive list of insertion editing sites became available.

Here we test the implementation of the frameshift DNA/protein alignment algorithm given in Fig. 1 for the task of annotating the *P. polycephalum* mitogenome (62,862 nt [36]) by homology search. We use the 67 protein sequences encoded by the mitochondrial genome of *Reclinomonas americana*. This jakobid excavate is only very distantly related to *P. polycephalum*. It has been reported, however, to host the most complete, bacterial-like protein complement of all mitogenomes investigated so far [37]. We therefore aligned each of the *Reclinomonas americana* proteins semi-globally against the mitogenome of *P. polycephalum*.

The *P. polycephalum* mitogenome contains 39 protein coding genes annotated in Redbase[4] [35], 32 of which have a homolog in *R. americana*. For 18 of these the best hit of the DNA-Protein alignment correctly identifies the genomic location of the gene in *P. polycephalum* with only minor deviations of the exact start and stop positions. In [33] 9 genes previously not identified in *P. polycephalum* were annotated with a handcrafted DNA-Protein alignment algorithm that specifically favours C insertions using all proteins in NCBI's non-redundant protein database.

Our approach recovers three of these nine proteins. However, we have made no efforts to optimize parameters, our algorithm does not distinguish between C insertions and other frame-shifting insertions or deletions, and we use only a single, evolutionary very remote mitogenome as query. We recover also nearly half of the C insertion sites. Using the *P. polycephalum* proteins as query, we find nearly all editing sites located in the coding regions (e.g., 76 of the 79 in nad5).

The purpose of this application example, however, is not to improve the annotation of the *P. polycephalum* mitogenome beyond the level of accuracy that was achieved with the help of transcriptome sequencing [35]. Our point is that a pilot study into this rather specialized topic can be set up literally within a few hours with the help of grammar products and the software support described in Section 2.6.

Applications such as the alignment of tens or more protein sequences to full genomes, albeit a rather small one in this case, require a modicum of performance considerations. Since we need to scan a full genome (though one only about 60,000 nt in length), we have opted for a sliding window approach for the DNA sequence, whereas the protein sequence is always aligned in full.

[4] http://bioserv.mps.ohio-state.edu/redbase/.

In addition, the algorithm makes use of multiple cores on a single machine in a parallel setup. This algorithm is embarrassingly parallel as all pairs of DNA windows and proteins can be aligned simultaneously, given enough resources.

The final ingredient to good performance is the implementation of the dynamic programming algorithm itself. Our implementation uses `ADPfusion` as the underlying dynamic programming framework. In this way, questions of performance for grammar products are reduced to those of better grammar design. `ADPfusion` then transforms the resulting grammar into efficient Haskell code.

Our larger-scale DNA-Protein example performs quite well—although we, of course, have no algorithm in `C` to compare against. The alignments of the 67 protein sequences of various lengths ranging from around 100 amino acids to several hundred to the approximately 60,000 nt of the mitogenome can be done in 289 minutes on an Intel Xeon E5-2680 running at 2.7 GHz running in single threaded mode.

3.3 Sankoff's Consensus Structure Algorithm

A classical problem in RNA bioinformatics is the simultaneous computation of a pairwise alignment and a consensus secondary structure of two input RNA sequences. David Sankoff already noticed in [12] that this problem smells of product structures.

For the sake of brevity we consider here only a variant of the "Nussinov grammar" [38] that distinguishes the non-terminal S for unconstrained structures and the non-terminal B for secondary structures enclosed by a base pair instead of the more commonly used "Zuker grammar" that accounts for the full loop decomposition [39]. The grammar $\mathcal{NUS}$ has the productions

$$\begin{aligned} &S \to \$ \mid Sa \mid SB \\ &B \to aS\hat{a}, \end{aligned} \tag{37}$$

where the terminals a and $\hat{a}$ denote nucleotides that can pair with each other. This is just a short hand for the six legal base pairs explicitly, i.e., $aS\hat{a} = \mathtt{a}S\mathtt{u} \mid \mathtt{c}S\mathtt{g} \mid \mathtt{g}S\mathtt{c} \mid \mathtt{g}S\mathtt{u} \mid \mathtt{u}S\mathtt{a} \mid \mathtt{u}S\mathrm{g}$.

The $*$-operation is easily generalized to arbitrary CFGs in the form $(A \to \alpha\beta\dots\gamma) * 2 = \binom{A}{A} \to \binom{\alpha}{\alpha}\binom{\beta}{\beta}\dots\binom{\gamma}{\gamma}$, where $\alpha, \beta, \dots, \gamma$ are either terminals or non-terminals. The natural version of the Sankoff algorithm for two input sequences is

$$\begin{aligned} &\binom{S}{S} \to \binom{\$}{\$} \mid \binom{S}{S}\binom{a}{a} \mid \binom{S}{S}\binom{a}{\varepsilon} \mid \binom{S}{S}\binom{\varepsilon}{a} \\ &\binom{S}{S} \to \binom{S}{S}\binom{B}{B} \\ &\binom{B}{B} \to \binom{a}{a}\binom{S}{S}\binom{\hat{a}}{\hat{a}}. \end{aligned} \tag{38}$$

This can be expressed much more compactly in the form

$$\mathcal{SANK} = \mathcal{NW} + (S \to SB,\ B \to aS\hat{a}) * 2, \tag{39}$$

when we allow base pairs in either sequence only when they also appear in the consensus. More complex grammars are required when we allow e.g. also the breaking of arcs, the insertion and deletion of entire base pairs, or alignments of paired and unpaired bases, see e.g., [7]. This begs the question whether such generalization could be obtained as subsets of the product of $\mathcal{NUS}$ with itself:

$$\begin{aligned} &\binom{S}{S} \to \binom{\$}{\$} \mid \binom{\$}{Sa} \mid \binom{\$}{SB} \mid \binom{Sa}{\$} \mid \binom{Sa}{Sa} \mid \binom{Sa}{SB} \mid \\ &\qquad \binom{SB}{\$} \mid \binom{SB}{Sa} \mid \binom{SB}{SB} \\ &\binom{S}{B} \to \binom{\$}{aS\hat{a}} \mid \binom{Sa}{aS\hat{a}} \mid \binom{SB}{aS\hat{a}} \\ &\binom{B}{S} \to \binom{aS\hat{a}}{\$} \mid \binom{aS\hat{a}}{Sa} \mid \binom{aS\hat{a}}{SB} \\ &\binom{B}{B} \to \binom{aS\hat{a}}{aS\hat{a}}. \end{aligned} \tag{40}$$

A closer inspection of this "formal" product reveals several problems.

While some of the formal right hand sides have natural explanations, such as $\binom{Sa}{Sa} \simeq \binom{S}{S}\binom{a}{a}$, others require the introduction of ε symbols, such as $\binom{aS\hat{a}}{Sa} \simeq \binom{a}{\varepsilon}\binom{S}{S}\binom{\hat{a}}{a}$. Other terms are more difficult to make sense of. For instance, what should we mean by $\binom{Sa}{SB}$? We might write $\binom{Sa}{SB} \simeq \binom{S}{S}\binom{a}{B}$ leaving us with an undesirable combination of a terminal and a non-terminal. In line with our construction for linear grammars we may include all order-preserving combinations in a form such as the following $\binom{Sa}{SB} \simeq \binom{S}{S}\binom{a}{\varepsilon}\binom{\varepsilon}{B} \mid \binom{S}{S}\binom{\varepsilon}{B}\binom{a}{\varepsilon}$ or possibly even more general expansions. In contrast to the linear grammars considered in the previous section general CFGs can have arbitrary strings of terminals and non-terminals as the r.h.s. of their productions. This may lead to an exponentially large number of "padded" terms in the interpretation of a formal product term. As a consequence it becomes very difficult to establish the algebraic properties of such a product.

A possible remedy comes from considering normal forms, of which several types have been explored in detail for CFGs. The two best known ones are the Chomsky normal form and the Greibach normal form [40], [41]. Both normal forms have been useful both in practise and as a theoretical device. We therefore explore here the possibility to construct direct products of context-free grammars in Greibach normal form as the two-GNF has the useful property of a r.h.s. with a single terminal followed by zero, one, or two non-terminal symbols. This property simplifies questions of alignment considerably.

3.4 A Product for Greibach Normal Forms

Every context free grammar that does not produce \$ can be transformed into an equivalent grammar with rules of the form

$$A \to aBC \mid bD \mid c \tag{41}$$

known as its Greibach normal form of order 2, or simply two-GNF. If the empty string is produced, the rule $S \to \$$ must be added, where S is the start symbol. We ignore this technicality for brevity of exposition. It is easily included if one allows \$ as a terminal symbol. It is however mostly a question of *semantics* if a grammar should consider empty input tapes legal input or not.

A natural product for grammars in two-GNF, which we denote by $\circledcirc$, can be obtained as follows: Terminals in the product are pairs of terminals of the input grammars, and the set of non-terminals is, as in the case of linear grammars, the Cartesian product of the sets of input non-terminals augmented by non-terminals of the form $\binom{A}{\varepsilon}$ and $\binom{\varepsilon}{A}$. The production rules of the product are the following:

$$\begin{aligned}
&(A_1 \to a_1 B_1 C_1) \circledcirc (A_2 \to a_2 B_2 C_2)\\
&= \left(\binom{A_1}{A_2} \to \binom{a_1}{a_2} \binom{B_1}{B_2} \binom{C_1}{C_2} \right)\\
&(A_1 \to a_1 B_1 C_1) \circledcirc (A_2 \to b_2 D_2)\\
&= \left(\binom{A_1}{A_2} \to \binom{a_1}{b_2} \binom{B_1}{D_2} \binom{C_1}{\varepsilon} \mid \binom{a_1}{b_2} \binom{B_1}{\varepsilon} \binom{C_1}{D_2} \right)\\
&(A_1 \to a_1 B_1 C_1) \circledcirc (A_2 \to c_2)\\
&= \left(\binom{A_1}{A_2} \to \binom{a_1}{c_2} \binom{B_1}{\varepsilon} \binom{C_1}{\varepsilon} \right)\\
&(A_1 \to b_1 D_1) \circledcirc (A_2 \to a_2 B_2 C_2)\\
&= \left(\binom{A_1}{A_2} \to \binom{b_1}{a_2} \binom{D_1}{B_2} \binom{\varepsilon}{C_2} \mid \binom{b_1}{a_2} \binom{\varepsilon}{B_2} \binom{D_1}{C_2} \right)\\
&(A_1 \to b_1 D_1) \circledcirc (A_2 \to b_2 D_2)\\
&= \Bigg(\binom{A_1}{A_2} \to \binom{b_1}{b_2} \binom{D_1}{D_2} \mid\\
&\qquad \boxed{\binom{b_1}{b_2} \binom{D_1}{\varepsilon} \binom{\varepsilon}{D_2} \mid \binom{b_1}{b_2} \binom{\varepsilon}{D_2} \binom{D_1}{\varepsilon}} \Bigg)\\
&(A_1 \to b_1 D_1) \circledcirc (A_2 \to c_2) = \left(\binom{A_1}{A_2} \to \binom{b_1}{c_2} \binom{D_1}{\varepsilon} \right)\\
&(A_1 \to c_1) \circledcirc (A_2 \to a_2 B_2 C_2)\\
&= \left(\binom{A_1}{A_2} \to \binom{c_1}{a_2} \binom{\varepsilon}{B_2} \binom{\varepsilon}{C_2} \right)\\
&(A_1 \to c_1) \circledcirc (A_2 \to b_2 D_2) = \left(\binom{A_1}{A_2} \to \binom{c_1}{b_2} \binom{\varepsilon}{D_2} \right)\\
&(A_1 \to c_1) \circledcirc (A_2 \to c_2) = \left(\binom{A_1}{A_2} \to \binom{c_1}{c_2} \right).
\end{aligned} \tag{42}$$

By construction, $\circledcirc$ is commutative (up to exchanging the coordinates). One easily checks that the product grammar is again in two-GNF since the r.h.s. of each production consists of a terminal followed by zero, one, or two non-terminal symbols. As in the case of the linear grammars we explain the productions of non-terminals of the form $\binom{A}{\varepsilon}$ by the productions of A in the first factor grammar, for instance $\binom{A}{\varepsilon} \to \binom{a}{\varepsilon}\binom{B}{\varepsilon}\binom{C}{\varepsilon}$. The distributive law $(P_1 + P_2) \circledcirc\ P_3 = P_1 \circledcirc P_3 + P_2 \circledcirc P_3$ also holds by construction.

To show that $\circledcirc$ is associative, it therefore suffices to show that the product is associative for any three productions. Since there are only three types of rules in a two-GNF, it suffices to consider the 27 possible products of triples of production rules, which altogether lead to 57 rules. We used a computational proof to establish that associativity is indeed satisfied, see Supplemental Material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TCBB.2014.2326155. It is important to note that the two "decoupling rules" for $(A_1 \to b_1 D_1) \circledcirc (A_2 \to b_2 D_2)$ indicated by the box in Eq. (42) are necessary for associativity of the product.

Linear grammars can be understood as special cases of two-GNF with productions of the form $A \to Bx \mid y$ (except that we now deal with right linear instead of left linear grammars). A comparison of the definition of $\otimes$ in Eq. (12) and $\circledcirc$ in Eq. (42) shows that the restriction of $\circledcirc$ to linear grammars does not recover (the mirror image of) $\otimes$. The discrepancy are exactly the two "decoupling rules" necessary for associativity of the $\circledcirc$ product. For instance, the $\circledcirc$-square of the step grammar $(X \to aX \mid \varepsilon X)$ has the productions

$$\begin{aligned}
\binom{X}{X} \to &\binom{a}{a}\binom{X}{X} \mid \binom{a}{a}\binom{X}{\varepsilon} \mid \binom{a}{a}\binom{\varepsilon}{X} \mid\\
&\binom{a}{\varepsilon}\binom{X}{X} \mid \binom{a}{\varepsilon}\binom{X}{\varepsilon} \mid \binom{a}{\varepsilon}\binom{\varepsilon}{X} \mid\\
&\binom{\varepsilon}{a}\binom{X}{X} \mid \binom{\varepsilon}{a}\binom{X}{\varepsilon} \mid \binom{\varepsilon}{a}\binom{\varepsilon}{X} \mid\\
&\binom{\varepsilon}{\varepsilon}\binom{X}{X} \mid \binom{\varepsilon}{\varepsilon}\binom{X}{\varepsilon} \mid \binom{\varepsilon}{\varepsilon}\binom{\varepsilon}{X}.
\end{aligned} \tag{43}$$

Only the first term in each line appears in the $\otimes$ product.

We have fully implemented definition of the $\circledcirc$ product for the two-GNF of Eq. (42), so that general CFGs in two-GNF can be defined and multiplied like linear (left-, right-, and general linear) grammars in our domain-specific language, providing access to an efficient implementation of the resulting multi-tape product grammars.

4 Discussion

Our main contribution is a formal, abstract algebra on linear grammars. This algebra provides operations to create complex, multi-tape grammars from simple, single-tape atomic ones. More informally, we have created a method and implementation to "multiply" dynamic programming algorithms. We also provide a compiler framework that makes the grammars readily available for actual deployment with good performance of the resulting code. Products of linear grammars make it very easy to construct the grammars underlying key algorithms in the field of string comparison starting from almost trivial single-tape factors. This approach is particularly fruitful in highly specialized applications as it drastically reduces the efforts required for implementing prototypes, as the example of DNA/protein alignments with frameshifts shows. The work presented here is just a first step towards a general theory of grammar products. Many questions, both theoretical and practical, remain open.

Although we have succeeded in constructing an algebraically meaningful product operation of CFGs in normal form it is currently restricted to the Greibach normal form. A fully generic version of the grammar product currently eludes us. While this poses no theoretical problem given that every CFG can be transformed into an equivalent CFG in (Greibach) Normal Form, it poses a problem in practice. Often, a production rule is associated with a certain structural feature that one wants to retain. Transformations into normal form also increase the number of non-terminals (and thereby resource usage) by a polynomial depending on the number of non-terminals [42].

It will also be important to explore both the interplay of different operators on grammars (especially our $+$ operation and the union ($\cup$) of grammars), and to formalize meaning and operation. This will provide, in the long-term, a full-fledged algebraic framework in which it should be easily possible to describe even complex grammatical problems. As pointed out by a reviewer, it may sometimes be convenient to introduce a grammar $G = (\{Z\}, \{\}, \{\})$ for the purpose of introducing the non-terminal Z (using $+$) or for the purpose of removing all rules that contain Z (using $-$). For instance, $(\{Y\}, \{\}, \{\}) * k$ could remove the unwanted double-deletion $\binom{Y}{Y}$ in Equ. (25) in a quite convenient manner. Our formalism, however, defines algebraic operations primarily on sets of production rules. This construction thus will require careful (re-)definition of the algebraic operations acting on grammars.

An intriguing idea (proposed by Robert Giegerich) is to identify irreducible grammars from which more complex ones can be constructed, possibly in a unique way. To that end, we will require a much better understanding of abstract algebras on grammars as well as the impact of constructions such as $(Z, \{\}, \{\})$. Since k-tape grammars can formally be written without making the dimension of the non-terminals explicit, a conceptually related question is whether product grammars can be efficiently recognized. More abstractly: ist there a "unique prime-factorization theorem" for grammars akin to analogous results for graphs? [43].

Another avenue of future research is the question of semantic ambiguity [44] of the resulting grammars. Simple products of the same grammar yield ambiguous alignments on sequences of in-dels, that is multiple derivations exist that yield the same alignment. This problem was studied extensively for stochastic context-free grammars implementing covariance models for structural RNA search [45]. It is typically dealt with a good grammar design that explicitly allows only one order of successive insertions and deletions on multiple tapes. Automatic dis-ambiguation is probably complicated but would further simplify the creation of complex multi-tape grammars.

In this contribution we have focussed entirely on the grammars underlying the dynamic programming algorithms and disregarded almost entirely the construction of scoring algebras for product grammars. We anticipate that in many cases, a scoring algebra can be expressed as a form of product itself where the two scoring functions (one for each grammar) are themselves combined in some well-defined form. One possibility is the use of a folding operation to combine scores for subsets of the individual dimensions. It then follows that given two algebras $\mathcal{A}_{G_1}$ and $\mathcal{A}_{G_2}$ for grammars G_1 and G_2 we should be able to define an operation $\mathcal{A}_{G_1} \otimes_\kappa \mathcal{A}_{G_2}$ which generates appropriate algebras from algebras for atomic grammars. As long as κ has some structure similar to a fold or another operation on subsets of the dimensions (of the grammars) involved, appropriate products can be automatically defined. This will become particularly useful when aiming at ADP-like [46] algebra-products to explore the rich space of combined algebras on grammars constructed from algebraic operations on atomic grammars.

In Section 3.1 on local alignments we mentioned that a naïve memoization of the three non-terminals of the Smith-Waterman algorithm leads to a three-fold increase in memory usage compared to the usual implementation based on one table and a neutral element in the scoring function. In case of local alignments, the evaluation functions attached to each production rule return a constant value, often the neutral element (i.e., a score of 0 for summations), for all productions and all substrings.

We plan to extend our framework to make it possible to evaluate combinations of grammars and algebras in more complex ways. This should allow us to automatically determine what kind of memo-table is required for each grammar and algebra, thereby optimizing memory consumption of the dynamic programming algorithms.

Good and optimal table designs based on yield size analysis have been considered in [47], extending earlier ideas on more restricted dynamic programming algorithms [48]. Our proposed extension will consider not only the yield size but the actual evaluation algebra, thereby including more domain-specific knowledge.

Acknowledgments

This work was funded, in part, by the Austrian FWF, project "SFB F43 RNA regulation of the transcriptome" and the German DFG project "MI439/14-1". CHzS thanks Jing, Katja, Lydia, and Nancy (and gin, as well as a mad man in a box). The authors thank Maria Walter for her hospitality that was very conducive for the development of this work, and the referees – Prof. Robert Giegerich and one anonymous – for providing in-depth comments.

References

[1] D. J. Lipman, S. F. Altschul, and J. D. Kececioglu, "A tool for multiple sequence alignment," *Proc. Nat. Acad. Sci. USA*, vol. 86, no. 12, pp. 4412–4415, 1989.

[2] R. Giegerich and C. Meyer, "Algebraic dynamic programming," in *Proc. Algebraic Methodol. Softw. Technol.*, 2002, vol. 2422, pp. 349–364.

[3] R. Giegerich, C. Meyer, and P. Steffen, "A discipline of dynamic programming over sequence data," *Sci. Comput. Program.*, vol. 51, no. 3, pp. 215–263, 2004.

[4] C. M. Reidys, F. W. D. Huang, J. E. Andersen, R. C. Penner, P. F. Stadler, and M. E. Nebel, "Topology and prediction of RNA pseudoknots," *Bioinformatics*, vol. 27, pp. 1076–1085, 2011.

[5] H. Chitsaz, R. Salari, S. C. Sahinalp, and R. Backofen, "A partition function algorithm for interacting nucleic acid strands," *Bioinformatics*, vol. 25, pp. i365–i373, 2009.

[6] F. W. D. Huang, J. Qin, C. M. Reidys, and P. F. Stadler, "Partition function and base pairing probabilities for RNA-RNA interaction prediction," *Bioinformatics*, vol. 25, pp. 2646–2654, 2009.

[7] S. Will, C. Schmiedl, M. Miladi, M. Möhl, and R. Backofen, "SPARSE: Quadratic time simultaneous alignment and folding of RNAs without sequence-based heuristics," in *Proc. 17th Int. Conf. Res. Comput. Mol. Biol.*, 2013, pp.289–290.

[8] C. Höner zuSiederdissen, I. L. Hofacker, and P. F. Stadler, "How to multiply dynamic programming algorithms," in *Proc. Brazilian Symp. Bioinformat.*, 2013, pp. 82–93.

[9] O. Gotoh, "An improved algorithm for matching biological sequences," *J. Mol. Biol.*, vol. 162, pp. 705–708, 1982.

[10] F. Lefebvre, *Grammairs s-attribuées multi-bandes et applications à l'analyse automatique de séquences biologiques*, Ph.D. dissertation, École Politechnique, Palaiseau, France, 1997.

[11] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *J. Mol. Biol.*, vol. 48, no. 3, pp. 443–453, 1970.

[12] D. Sankoff, "Simultaneous solution of the RNA folding, alignment and protosequence problems," *SIAM J. Appl. Math.*, vol. 45, pp. 810–825, 1985.

[13] The GHC Team. (1989–2013). The Glasgow Haskell Compiler (GHC) [Online]. Available: http://www.haskell.org/ghc/

[14] T. Sheard, and S. P. Jones, "Template Meta-programming for Haskell," in *Proc. ACM SIGPLAN Workshop Haskell*, 2002, pp. 1–16.

[15] G. Mainland, "Why it's nice to be quoted: Quasiquoting for Haskell," in *Proc. ACM SIGPLAN Workshop Haskell Workshop*, 2007, pp. 73–82.

[16] D. Coutts, R. Leshchinskiy, and D. Stewart, "Stream fusion: From lists to streams to nothing at all," in *Proc. 12th ACM SIGPLAN Int. Conf. Funct. Program.*, 2007, pp. 315–326.

[17] C. Höner zu Siederdissen, "Sneaking around concatMap: Efficient combinators for dynamic programming," in *Proc. 17th ACM SIGPLAN Int. Conf. Funct. Program.*, 2012, pp. 215–226.

[18] G. Mainland, R. Leshchinskiy, S. P. Jones, and S. Marlow, "Exploiting vector instructions with generalized stream fusion," in *Proc. 18th ACM SIGPLAN Int. Conf. Funct. Program.*, 2013, pp. 37–48.

[19] G. Keller, M. M. T. Chakravarty, R. Leshchinskiy, S. Peyton Jones, and B. Lippmeier, "Regular, shape-polymorphic, parallel arrays in Haskell," in *Proc. 15th ACM SIGPLAN Int. Conf. Funct. Program.*, 2010, pp. 261–272.

[20] O. Gotoh, "Alignment of three biological sequences with an efficient traceback procedure," *J. Theor. Biol.*, vol. 121, pp. 327–337, 1986.

[21] T. G. Dewey, "A sequence alignment algorithm with an arbitrary gap penalty function," *J. Comput. Biol.*, vol. 8, pp. 177–190, 2001.

[22] A. S. Konagurthu, J. Whisstock, and P. J. Stuckey, "Progressive multiple alignment using sequence triplet optimization and three-residue exchange costs," *J. Bioinf. Comput. Biol.*, vol. 2, pp. 719–745, 2004.

[23] M. Kruspe, and P. F. Stadler, "Progressive multiple sequence alignments from triplets," *BMC Bioinformat.*, vol. 8, p. 254, 2007.

[24] L. Steiner, P. F. Stadler, and M. Cysouw, "A pipeline for computational historical linguistics," *Lang. Dynamics Change*, vol. 1, pp. 89–127, 2011.

[25] N. Retzlaff, "Bigramm-Alignierung und ihre Anwendung in der historischen Linguistik," Bachelors Thesis, Department of Computer Science, Eberhard Karls Universität Tbingen, Tbüingen, Germany, 2013.

[26] T. F. Smith, and M. S. Waterman, "Identification of common molecular subsequences," *J. Mol. Biol.*, vol. 147, pp. 195–197, 1981.

[27] S. R. Eddy, "HMMER: Profile HMMs for protein sequence analysis," *Bioinformatics*, vol. 14, pp. 755–763, 1998.

[28] S. R. Eddy, and R. Durbin, "RNA sequence analysis using covariance models," *Nucl. Acids Res.*, vol. 22, pp. 2079–2088, 1994.

[29] E. P. Nawrocki, D. L. Kolbe, and S. R. Eddy, "Infernal 1.0: inference of RNA alignments," *Bioinformatics*, vol. 25, no. 10, pp. 1335–1337, 2009.

[30] F. Lefebvre, "An optimized parsing algorithm well suited to RNA folding," in *Proc. 3rd Int. Conf. Intell. Syst. Mol. Biol.*, 1995, pp. 222–230.

[31] K. Prüfer, U. Stenzel, M. Hofreiter, S. Pääbo, J. Kelso, and R. E. Green, "Computational challenges in the analysis of ancient DNA," *Genome Biol.*, vol. 11, p. R47, 2010.

[32] J. M. Gott, N. Parimi, and R. Bundschuh, "Discovery of new genes and deletion editing in *Physarum* mitochondria enabled by a novel algorithm for finding edited mRNAs," *Nucl. Acids Res.*, vol. 33, pp. 5063–5072, 2005.

[33] C. Beargie, T. Liu, M. Corriveau, H. Y. Lee, J. Gott, and R. Bundschuh, "Genome annotation in the presence of insertional RNA editing," *Bioinformatics*, vol. 24, pp. 2571–2578, 2008.

[34] R. Bundschuh, "Computational approaches to insertional RNA editing," *Methods Enzymol.*, vol. 424, pp. 173–195, 2007.

[35] R. Bundschuh, J. Altmüller, C. Becker, P. Nürnberg, and J. M. Gott, "Complete characterization of the edited transcriptome of the mitochondrion of *Physarum polycephalum* using deep sequencing of RNA," *Nucl. Acids Res.*, vol. 39, pp. 6044–6055, 2011.

[36] H. Takano, T. Abe, R. Sakurai, Y. Moriyama, Y. Miyazawa, H. Nozaki, S. Kawano, N. Sasaki, and T. Kuroiwa, "The complete DNA sequence of the mitochondrial genome of *Physarum polycephalum*," *Mol. General Genetics*, vol. 264, pp. 539–545, 2001.

[37] B. F. Lang, G. Burger, C. J. O'Kelly, R. Cedergren, G. B. Golding, C. Lemieux, D. Sankoff, M. Turmel, and M. W. Gray, "An ancestral mitochondrial DNA resembling a eubacterial genome in miniature," *Nature*, vol. 387, pp. 493–497, 1997.

[38] R. Nussinov, G. Piecznik, J. R. Griggs, and D. J. Kleitman, "Algorithms for loop matching," *SIAM J. Appl. Math.*, vol. 35, pp. 68–82, 1978.

[39] R. D. Dowell, and S. R. Eddy, "Evaluation of several lightweight stochastic context-free grammars for RNA secondary structure prediction," *BMC Bioinformat.*, vol. 5, p. 71, 2004.

[40] S. A. Greibach, "A new normal-form theorem for context-free phrase structure grammars," *J. ACM*, vol. 12, pp. 42–52, 1965.

[41] A. Ehrenfeucht, and G. Rozenberg, "An easy proof of Greibach normal form," *Inf. Control*, vol. 63, pp. 190–199, 1984.

[42] N. Blum, and R. Koch, "Greibach normal form transformation revisited," *Inf. Comput.*, vol. 150, pp. 112–118, 1999.

[43] R. Hammack, W. Imrich, and S. Klavžar, *Handbook of Product Graphs*, 2nd ed. Boca Raton, FL, USA: CRC Press, 2011.

[44] R. Giegerich, "Explaining and controlling ambiguity in dynamic programming," in *Combinatorial Pattern Matching*. New York, NY, USA: Springer, 2000, pp. 46–59.

[45] R. Giegerich, and C. Höner zu Siederdissen, "Semantics and ambiguity of stochastic RNA family models," *IEEE/ACM Trans. Comput. Biol. Bioinformat.*, vol. 8, no. 2, pp. 499–516, Mar. 2011.

[46] P. Steffen, and R. Giegerich, "Versatile and declarative dynamic programming using pair algebras," *BMC Bioinformat.*, vol. 6, p. 224, 2005.

[47] P. Steffen and R. Giegerich, "Table design in dynamic programming," *Inf. Comput.*, vol. 204, no. 9, pp. 1325–1345, 2006.

[48] H. L. Bodlaender and J. A. Telle. (2004, Dec.). Space-efficient construction variants of dynamic programming. *Nordic J. Comput.* [Online]. *11(4)*, pp. 374–385. Available: http://dl.acm.org/citation.cfm?id=1062991.1062995

Christian Höner zu Siederdissen is a postdoctoral researcher with the Institute for Theoretical Chemistry at the University of Vienna. His research interests include optimization problems in biology, Bayesian statistics, formal languages, and functional programming.

Ivo L. Hofacker is a professor at the Departments of Chemistry and of Computer Science at the University of Vienna, where he heads the Institute for Theoretical Chemistry and the Research Group Bioinformatics and Computational Biology, respectively. His research interests focus on the prediction of RNA structures, noncoding RNA annotation, and the analysis of folding landscapes.

Peter F. Stadler received the PhD degree in Chemistry from the University of Vienna in 1990, and then worked as an assistant and associate professor for theoretical chemistry at the same School. In 2002, he moved to Leipzig as a full professor for bioinformatics. Since 1994, he is an external professor at the Santa Fe Institute. He is an external scientific member of the Max Planck Society since 2009 and a corresponding member abroad of the Austrian Academy of Sciences since 2010.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.