Modelling programmed frameshifting with frameshift machines. *

Mark Daley[1], Ian McQuillan[2,†]

[1] Department of Computer Science, Department of Biology
University of Western Ontario
London, ON, N6A 5B7, Canada

[2] Department of Computer Science
University of Saskatchewan
Saskatoon, SK, S7N 5A9, Canada

Abstract

The translation of a messenger RNA into a functional protein is one of the most fundamental molecular processes in a cell. Groups of three ribonucleotides, called codons, uniquely specify amino acids to be used in the construction of a protein. When the translation process skips a number of bases it is possible for the reading frame of the RNA to be shifted. By making use of multiple reading frames, organisms and viruses are able to encode multiple proteins in a single gene. We propose here a formal model of these frameshifting events and investigate its basic mathematical properties and their relevance to biological systems. In addition, multiple time-efficient algorithms are created for use in the study of frameshifting. Some of these algorithms are created to work in general, for any type of frameshifting which could be found in organisms, while others are optimized for known specialized types of frameshifting.

Key words: bioinformatics, frameshifting, formal language theory, genetic code, mathematical modelling

Introduction

The fundamental dogma of molecular biology is that of a strand of DNA, encoding a gene, transcribed into a messenger RNA which is then translated into a string of amino acids which comprises the finished product: a protein. While this model is valid in a high-level sense, modern biological discoveries have revealed that the process is much more complex, at every stage, than originally thought. Indeed, DNA can be edited pre-transcription [1], several classes of post-transcriptional RNA editing are now known, for example, intron removal and uridine insertion and deletion in plants and kinetoplastids [2], as well as post-translational editing and assembly of proteins.

This complex processing of genetic information allows for a huge diversity of proteins (e.g. a large proteome) from a relatively smaller genome; indeed, the human genome itself is estimated to contain only 20000-25000 genes while *C. elegans*, a microscopic nematode of around 1000 cells, has already 19,000 genes. Clearly, a large difference in the observed complexity of *C. elegans* and

*Published in Natural Computing, 9, 239–261, 2010. https://doi.org/10.1007/s11047-009-9144-x
†Corresponding author, mcquillan@cs.usask.ca, ph: +306-966-2900, fax: +306-966-4884.

humans must be related to the complexity introduced by the transformation of genetic information rather than simply the number of genes.

In this paper, we will formally model and study one such transformational process: that of frameshifting during the translation of RNA into a protein. Given the (perhaps edited) RNA transcript of a gene, the process of translation begins with the small ribosomal subunit binding upstream of a particular sequence of three ribonucleotides called a *start codon*. The process of translation proceeds linearly beginning with the start codon [3], with each substring of three ribonucleotides, called a *codon*, indicating a single unique amino acid that should be placed next in the construction of the protein. Note that the protein is also constructed in a linear fashion though, once translation is finished and it is free of the ribosome, the protein will fold to form a three dimensional structure. When a special type of codon, known as a stop codon, is reached, translation halts and the protein is released into the cytoplasm.

Thus, the process of translation consists of the relatively straightforward mapping of triples of ribonucleotides on to single amino acids. We can see immediately, however, that if for some reason, we were to skip a single ribonucleotide, we would not only have a single-point error in our translation, but every codon downstream from the shift would be affected. This is referred to as a change of *reading frame* or *frameshift*. While it may seem that events of this nature are deleterious, this is often not strictly the case in practice. Many organisms make use of intentional frameshifting for purposes ranging from allowing for the simple production of multiple proteins from a single gene, to a crude form of regulating gene expression in viruses. See, for example, the single-gene encoding of the *gag* and *gag-pol* proteins in retroviruses [4]; the production of either *gag* or *gag-pol* during translation depends on frameshift events which occur with a relatively fixed frequency, resulting in a global production of different amounts of the two products.

The exact mechanism of frameshifting varies from case-to-case and organism-to-organism. Most significantly for our purposes, in real systems, frameshift events may only take place at a subset of codons which are flanked by very specific contexts. For a good survey of such frameshifting mechanisms, organisms and sites at which this can occur, we direct the reader to [5] and [6].

This paper presents the *frameshift machine* as a formal model of the frameshifting process. In section 2, we discuss mathematical notation and preliminaries. In section 3, we give the definition of a frameshift machine and define some abstract properties that will be useful to discuss them. In section 4, we will demonstrate some basic mathematical properties and construct a specialized algorithm for certain realistic types of frameshifting. In section 5, we create a suite of algorithms that work, in general, for different types of frameshifting, and in section 6 we present our conclusions.

The work in this paper differs from previous bioinformatics works on frameshifting. Most algorithms developed in the past compare homologous proteins or DNA with proteins that differ via frameshifting [7, 8, 9]. The algorithms developed here can be considered "exact algorithms" in that they do not compare sequences for similarity, but rather, compare sequences that differ by programmed frameshifting alone. Of particular interest are Propositions 0.4 and 0.5 which use an algorithm that compares protein segments that differ via a single frameshift (using realistic types of frameshifting) in linear time, even scanning the proteins in a unidirectional manner. Moreover, they can produce regular expressions for exactly the RNAs that would produce these protein segments in the same time as well. Algorithm 3, producing the regular expression can be seen as an analogue to the study of synonymous/non-synonymous mutations when only one protein is produced from an mRNA. In that case, the characterization is quite simple, as one can just examine the genetic code for mutations which do not change the amino acids. But in the case where two (or more) proteins are produced, a more complex algorithm (Algorithm 3) is necessary in order to understand the synonymous mutations. Hence, exact algorithms are of definite interest. Furthermore, by abstracting the frameshifting process in terms of a so-called "frameshift machine", we are able

to analyze the process mathematically, and even construct algorithms regardless of the type of frameshifting used.

Mathematical preliminaries

We will start by defining the mathematical notation needed in order to define and reason about frameshift machines.

Let $\mathbb{Z}$ be the set of integers, let $\mathbb{N}$ be the set of positive integers and let $\mathbb{N}_0$ be the set of nonnegative integers. For $n \in \mathbb{N}_0$, let $\mathbb{Z}(n) = \{-n, \ldots, n\}$, $\mathbb{N}(n) = \{1, \ldots, n\}$, $\mathbb{N}_0(n) = \{0, \ldots, n\}$ and $\mathbb{N}_0^-(n) = \{-n, \ldots, 0\}$.

We refer to [10] for formal language theory preliminaries. Let Σ be a finite alphabet, which is simply a set of abstract symbols, such as the alphabet of nucleotides. We denote, by Σ^* and Σ^+, the sets of all words and non-empty words, respectively, over Σ and the empty word by λ. A language L is any subset of Σ^*. Thus, a language is any set of words over some alphabet. Let $x \in \Sigma^*$. We let $|x|$ denote the length of x. For $i \in \mathbb{N}$, let $x(i)$ be a_i if $x = a_1 \cdots a_i \cdots a_n$, $a_j \in \Sigma, 1 \leq j \leq n$, and undefined otherwise. For $i, \tau \in \mathbb{N}$, $x = a_1 \cdots a_n$, let $x(i, \tau) = a_i \cdots a_{i+\tau-1}$ if $a_i, \ldots, a_{i+\tau-1}$ are defined and undefined otherwise. For $i \in \mathbb{N}_0(|x|)$, let $x^{\leftarrow}(i)$ be λ if $i = 0$ and $x(1) \cdots x(i)$ otherwise. For $i \in \mathbb{N}(|x| + 1)$, let $x^{\rightarrow}(i)$ be λ if $i = |x| + 1$ and $x(i) \cdots x(|x|)$ otherwise.

For $n \in \mathbb{N}_0$, let $\Sigma^n = \{w \in \Sigma^* \mid |w| = n\}$, $\Sigma^{\geq n} = \{w \in \Sigma^* \mid |w| \geq n\}$ and $\Sigma^{\leq n} = \{w \in \Sigma^* \mid |w| \leq n\}$.

Let $x, y \in \Sigma^*$. We say x is a prefix of y, written $x \leq_{\mathrm{p}} y$, if $y = xu$, for some $u \in \Sigma^*$. We say x is a suffix of y, written $x \leq_{\mathrm{s}} y$, if $y = ux$, for some $u \in \Sigma^*$.

We will also talk about both regular expressions and finite automata, defined in [10].

Frameshifting machines

We begin by defining a type of abstract machine which generalizes the biological process of translation while still allowing for shifts of reading frame. We will construct specific examples below. Then, we define some abstract properties which we use to analyze frameshift machines.

Definition 0.1 *A* frameshift machine *is a five-tuple $M = (\Sigma, \Gamma, \tau, E, \delta)$ where*

Σ *is the finite input alphabet,*
Γ *is the finite output alphabet,*
$\tau \in \mathbb{N}$ *is the frame size,*
$E \subseteq \Sigma^\tau$ *is the set of end frames,*
$\delta \subseteq \Sigma^* \times \Sigma^\tau \times \Sigma^* \times \Gamma \times \mathbb{Z}(\tau - 1)$,
δ *finite, is the transition relation.*

For a frameshift machine $M = (\Sigma, \Gamma, \tau, E, \delta)$ and a transition $(u, w, v, a, i) \in \delta$, we call u the left context*, v the* right context*, w the* input*, a the* output*, and i the* shift amount*.*

For a frameshift machine $M = (\Sigma, \Gamma, \tau, E, \delta)$, we define the derivation relation $\vdash_M$ on $\Sigma^* \times \Gamma^* \times \mathbb{N}$ by

$$(w, \alpha, n) \vdash_M (w, \alpha a, n + m + \tau)$$

if and only if $(u, w(n, \tau), v, a, m) \in \delta$ and $u \leq_{\mathrm{s}} w^{\leftarrow}(n - 1), v \leq_{\mathrm{p}} w^{\rightarrow}(n + \tau)$ where $w, u, v \in \Sigma^*, \alpha \in \Gamma^*, n, n + m + \tau \in \mathbb{N}(|w|), a \in \Gamma, m \in \mathbb{Z}$. Then, let $\vdash_M^*$ be the reflexive, transitive closure of $\vdash_M$.

	U	C	A	G	
U	UUU (f)	UCU (s)	UAU (y)	UGU (c)	U
	UUC (f)	UCC (s)	UAC (y)	UGC (c)	C
	UUA (l)	UCA (s)	UAA ($)	UGA ($)	A
	UUG (l)	UCG (s)	UAG ($)	UGG (w)	G
C	CUU (l)	CCU (p)	CAU (h)	CGU (r)	U
	CUC (l)	CCC (p)	CAC (h)	CGC (r)	C
	CUA (l)	CCA (p)	CAA (q)	CGA (r)	A
	CUG (l)	CCG (p)	CAG (q)	CGG (r)	G
A	AUU (i)	ACU (t)	AAU (n)	AGU (s)	U
	AUC (i)	ACC (t)	AAC (n)	AGC (s)	C
	AUA (i)	ACA (t)	AAA (k)	AGA (r)	A
	AUG (m)	ACG (t)	AAG (k)	AGG (r)	G
G	GUU (v)	GCU (a)	GAU (d)	GGU (g)	U
	GUC (v)	GCC (a)	GAC (d)	GGC (g)	C
	GUA (v)	GCA (a)	GAA (e)	GGA (g)	A
	GUG (v)	GCG (a)	GAG (e)	GGG (g)	G

Table 1: The genetic code associating codons with amino acids, using the $ character for stop codons [3].

Further, for $w \in \Sigma^*$ *and* $L \subseteq \Sigma^*$*, let* $M(w) = \{\alpha \mid (w, \lambda, 1) \vdash_M^* (w, \alpha, k), k \in \mathbb{N}(|w|), w(k, \tau) \in E\}$ *and* $M(L) = \bigcup_{w \in L} M(w)$*. Also, for* $\alpha \in \Gamma^*$ *and* $L \subseteq \Gamma^*$*, let* $M^{-1}(\alpha) = \{w \mid \alpha \in M(w)\}$ *and* $M^{-1}(L) = \bigcup_{\alpha \in L} M^{-1}(\alpha)$*.*

Although the type of machine described above is general enough to work over arbitrary finite alphabets and frame sizes, we are primarily concerned here with the biological case. As such, we fix the following notational conventions for the remainder of the paper. We use the alphabets $\Sigma_{\text{RNA}} = \{\text{U}, \text{C}, \text{A}, \text{G}\}$, $\Gamma_{\text{AA}} = \{\underline{\text{a}}, \underline{\text{r}}, \underline{\text{n}}, \underline{\text{d}}, \underline{\text{c}}, \underline{\text{e}}, \underline{\text{q}}, \underline{\text{g}}, \underline{\text{h}}, \underline{\text{i}}, \underline{\text{l}}, \underline{\text{k}}, \underline{\text{m}}, \underline{\text{f}}, \underline{\text{p}}, \underline{\text{s}}, \underline{\text{t}}, \underline{\text{w}}, \underline{\text{y}}, \underline{\text{v}}\}$ for the set of ribonucleotides and amino acids, respectively. We use the underlines to disambiguate between individual amino acids and variables, however we will omit the underlines in situations where it is clear from the context. We also define the partial function which maps codons onto amino acids, $\phi_{\text{GEN}} : \Sigma_{\text{RNA}}^3 \mapsto \Gamma_{\text{AA}}$, as done in the genetic code (see Table 1). Also, we define the set of stop codons, $E_{\text{SP}} = \{\text{UAA}, \text{UAG}, \text{UGA}\}$. We would like to note that the formal model could be used with alternate genetic codes (which are known to occur in some organisms [12]) or stop codons coding for amino acids [12], including the incorporation of the 21^{st} amino acid, selenocysteine [5].

Then, usual translation (without frameshifting) is just a special case of the following machine, $M_{\text{GEN}} = (\Sigma_{\text{RNA}}, \Gamma_{\text{AA}}, 3, E_{\text{SP}}, \delta_{\text{GEN}})$ where $\delta_{\text{GEN}} = \{(\lambda, w, \lambda, a, 0) \mid \phi_{\text{GEN}}(w) = a\}$.

The derivation relation describes how the inputs can be converted to outputs in a single step. If we have $(w, \alpha, n) \vdash_M (w, \alpha a, n + m + \tau)$, then w is the input (the RNA), α is the output (the protein segment) we have generated so far, we are currently scanning the n^{th} character of w, and there is a transition $(u, w(n, \tau), v, a, m)$ which implies that the next τ letters of w is $w(n, \tau)$, the machine outputs the letter a, and moves forward the codon size τ plus the frameshift amount m. As an example,

$$(\text{AUGGCCCGA}, \lambda, 1) \vdash_{M_{\text{GEN}}} (\text{AUGGCCCGA}, \underline{\text{m}}, 4) \vdash_{M_{\text{GEN}}} (\text{AUGGCCCGA}, \underline{\text{ma}}, 7).$$

Lastly, everything before the current position of the input must end in u, and everything after the

codon of the input must begin with v which in the case of M_{GEN} is always true since the left and right contexts are always empty. The contexts are necessary however as often times a stimulatory signal in the RNA distinct from the shift site is necessary in order to shift frames [5]. We will discuss the contexts more shortly. Then, $\vdash^*_{\text{GEN}}$ describes how inputs can be converted to outputs after arbitrarily many steps.

Then if w is a gene, then $M(w)$ would be the corresponding protein (or in the case where M can frameshift, a set of proteins). Similarly, a set of genes L generates a set of proteins $M(L)$. Conversely, if we have a protein α, then $M^{-1}(\alpha)$ is the set of RNAs which can generate α, and this is extended to sets of proteins.

Next, we introduce some properties that we will use to analyze frameshift machines.

Definition 0.2 *We call M* functional *if, for every $w \in \Sigma^\tau$ and $(u, w, v, a, i), (u', w, v', b, j) \in \delta$ implies $a = b$ for $a, b \in \Gamma, i, j \in \mathbb{Z}, u, v, u', v' \in \Sigma^*$. If M is functional, then we define $\delta(w) = a$ where $(u, w, v, a, i) \in \delta$ for some $w \in \Sigma^\tau, u, v \in \Sigma^*, a \in \Gamma, i \in \mathbb{Z}$. We call M* injective *if, for every $a \in \Gamma, (u, w, v, a, i), (u', y, v', a, j) \in \delta$ implies $w = y$ for $w, y \in \Sigma^\tau, i, j \in \mathbb{Z}, u, v, u', v' \in \Sigma^*$. We call M deterministic if, for every $w \in \Sigma^\tau, (u, w, v,\ a, i), (u', w, v', b, j) \in \delta$ implies both $a = b$ and $i = j$ for $a, b \in \Gamma, i, j \in \mathbb{Z}, u, v, u', v' \in \Sigma^*$. We call M nondeterministic otherwise.*

Furthermore, we call M a downstream frameshift machine *if $\delta \subseteq \Sigma^* \times \Sigma^\tau \times \Sigma^* \times \Gamma \times \mathbb{N}_0(\tau - 1)$ and we call M an* upstream frameshift machine *if $\delta \subseteq \Sigma^* \times \Sigma^\tau \times \Sigma^* \times \Gamma \times \mathbb{N}_0^-(\tau - 1)$. For $A \subseteq \mathbb{Z}(\tau - 1)$, we call M an A-frameshift machine, if $\delta \subseteq \Sigma^* \times \Sigma^\tau \times \Sigma^* \times \Gamma \times A$. We call M* non-contextual *if $\delta \subseteq \{\lambda\} \times \Sigma^\tau \times \{\lambda\} \times \Gamma \times \mathbb{Z}(\tau - 1)$. In the case where M is non-contextual, we also write transitions of δ as being ordered triples, where we leave out the left and right contexts and $\delta \subseteq \Sigma^\tau \times \Gamma \times \mathbb{Z}(\tau - 1)$.*

So, M_{GEN} is a frameshift machine which is non-contextual since an amino acid is only determined by a codon and not any other nucleotides to the left or right. It is also functional since each individual codon only has a single amino acid associated with it in the genetic code. Lastly, M_{GEN} is deterministic since codons uniquely determine an amino acid and it always shifts forward three letters (0-frameshifting). However, if we allow frameshifting, then our frameshift machine is still functional, however, we get nondeterminism. This would be obtained by adding extra transitions to M_{GEN}.

To study realistic extensions of M_{GEN} which allow frameshifting, we define the following:

Definition 0.3 *We say that a frameshifting machine $M = (\Sigma, \Gamma, \tau, E, \delta)$ is* extended *if $\Sigma = \Sigma_{\text{RNA}}, \Gamma = \Gamma_{\text{AA}}, \tau = 3, E_{\text{SP}} = E, \delta_{\text{GEN}} \subseteq \delta$ and M is functional.*

Then a machine M is extended if and only if M can do everything M_{GEN} can do, can potentially also frameshift, however can still output only the same amino acid given a codon. Most known types of biological frameshifting are extended and most are either a $\{0, 1\}$- or $\{0, -1\}$-machine. For example, in the prfB gene of *Escherichia coli* (reviewed in [5]), the sequence CUUUGAC can either terminate with the stop codon UGA, or shift forward one ribonucleotide. Thus, it would be modelled with an extended frameshift machine with an extra transition $(\lambda, \text{CUU}, \text{UGAC}, \underline{\text{l}}, 1)$. Thus, if we are scanning an RNA sequence and starting at the current position, the sequence CUUUGAC appears, then we can either use the "standard" transition and output $\underline{\text{l}}$ and then hit a stop codon thereby terminating, or the machine can use the new transition, which outputs $\underline{\text{l}}$, and then shifts forward so that the machine is scanning G and would likely next output $\underline{\text{d}}$. In this case, CUU becomes the input and UGAC the right context which is necessary to be directly after the input in order for frameshifting to occur. This example, along with many others in [5] show why the contexts are necessary.

Some of the results of this paper could be reformulated in terms of shift spaces, higher block shifts and higher power shifts (see [11]). These constructs allow to scan words one block at a time,

and then shift forward by a constant amount, while outputting (the constructs are much more general than only that purpose). In particular, it is possible to reformulate Proposition 0.1 in terms of these notions. However, this formulation, as defined, does not accommodate different shifting amounts, which is present whenever frameshifting occurs and is important for the General Algorithms section. Also, we wanted a model that more closely maps on to the biological mechanisms and nomenclature of frameshifting.

Properties and algorithms for extended machines

In this section, we will start by showing some interesting mathematical properties of extended machines, which are of particular interest. In addition, we will develop some specialized algorithms which will work only on these types of frameshift machines.

We note the following remark for arbitrary frameshift machines, relating the various mathematical properties, which follow directly from the definitions.

Remark 0.1 *Let $M = (\Sigma, \Gamma, \tau, E, \delta)$ be a frameshift machine. Then the following are true:*

1. *If M is deterministic, then M is functional,*
2. *If $w \in M^{-1}(L)$, for some $L \subseteq \Gamma^*$, then $w\Sigma^* \subseteq M^{-1}(L)$.*

Next, we use frameshift machines to examine an interesting property of the genetic code. Essentially, it shows that if translation is forced to proceed one ribonucleotide at a time, then there is only one RNA which could code for (except for the first and last nucleotides) each protein. It examines combinatorially the various ways that codons can overlap with each other. We will also discuss the relevance of this result below.

Let $M = (\Sigma, \Gamma, \tau, E, \delta)$ be a frameshift machine. We define $M^{\leftarrow} = (\Sigma, \Gamma, \tau, E, \delta')$ to be the frameshift machine where, for every $(u, w, v, a, i) \in \delta$, we let $(u, w, v, a, 1 - \tau) \in \delta'$. Then, at every step of the derivation relation, we are moving forward exactly one symbol. It is clear that this type of frameshift machine is unrealistic, although it may be of use, as we will see below.

Remark 0.2 *If M is functional, then $M^{\leftarrow}$ is functional.*

The injective condition, as defined, is too strong. As such, we now define a weakened version. Let $M = (\Sigma, \Gamma, \tau, E, \delta)$ be a frameshift machine. Let $k \in \mathbb{N}$. We say that M is k-pseudo-injective, if, for every $x_1, x_2 \in \Gamma^k, a \in \Gamma$,

$$(w, \lambda, k_0) \vdash_M^* (w, x_1, k_1) \vdash_M (w, x_1 a, k_2) \vdash_M^* (w, x_1 a x_2, k_3),$$
$$(v, \lambda, l_0) \vdash_M^* (v, x_1, l_1) \vdash_M (v, x_1 a, l_2) \vdash_M^* (v, x_1 a x_2, l_3),$$

implies $w(k_1, \tau) = v(l_1, \tau)$.

Roughly, M is k-pseudo injective if and only if, given $x_1 a x_2$ with x_1, x_2 of length k and a of length 1, there is only one possible input word of length τ which gets "translated" into a. This is a slightly weaker condition than injectivity, but it has similar side effects. The following two lemmata are quite technical, however they are used in the proposition below.

Lemma 0.1 *Let $M = (\Sigma, \Gamma, \tau, E, \delta)$ be an upstream k-pseudo-injective frameshift machine, let $x \in \Gamma^{\geq 2k+1}$ and let $x = x_1 x' a x_2$ with $x_1, x_2 \in \Gamma^k, a \in \Gamma, x' \in \Gamma^*$. Then*

$$(w, \lambda, k_0) \vdash_M^* (w, x_1, k_1) \vdash_M^* (w, x_1 x', k_2) \vdash_M (w, x_1 x' a, k_3) \vdash_M^* (w, x_1 x' a x_2, k_4),$$
$$(v, \lambda, l_0) \vdash_M^* (v, x_1, l_1) \vdash_M^* (v, x_1 x', l_2) \vdash_M (v, x_1 x' a, l_3) \vdash_M^* (v, x_1 x' a x_2, l_4),$$

implies $w(k_1) \cdots w(k_2 + \tau - 1) = v(l_1) \cdots v(l_2 + \tau - 1)$.

Proof. Assume

$$(w, \lambda, k_0) \vdash_M^* (w, x_1, k_1) \vdash_M (w, x_1a_1, k_2) \vdash_M \cdots$$
$$\vdash (w, x_1a_1 \cdots a_n, k_n) \vdash_M^* (w, x_1a_1 \cdots a_nx_2, k_{n+1}),$$

and

$$(v, \lambda, l_0) \vdash_M^* (v, x_1, l_1) \vdash_M (v, x_1a_1, l_2) \vdash_M \cdots$$
$$\vdash (v, x_1a_1 \cdots a_n, l_n) \vdash_M^* (v, x_1a_1 \cdots a_nx_2, l_{n+1}).$$

Then, $k_{i+1} \leq k_i + \tau$ and $l_{i+1} \leq l_i + \tau$, for all i, $1 \leq i < n$ since M is upstream. Also, $w(k_1, \tau) = v(l_1, \tau), \ldots, w(k_{n-1}, \tau) = v(l_{n-1}, \tau)$ since M is k-pseudo-injective. Hence, $w(k_1) \cdots w(k_{n-1}+\tau-1) = v(l_1) \cdots\ v(l_{n-1} + \tau - 1)$. ■

Lemma 0.2 *$M_{\text{GEN}}^{\leftarrow}$ is 1-pseudo-injective.*

Proof. We first note by examining the genetic code that given $a \in \Gamma_{\text{AA}}$ with $a \neq \underline{\text{s}}, \phi_{\text{GEN}}(a_1a_2a_3) = a, \phi_{\text{GEN}}(b_1b_2b_3) = a$, $a_1, a_2, a_3, b_1, b_2, b_3 \in \Sigma_{\text{RNA}}$ implies $a_2 = b_2$.

Let $M = M_{\text{GEN}}^{\leftarrow}$. Let $w, v \in \Sigma_{\text{RNA}}^*, a, b, c \in \Gamma_{\text{AA}}$. Assume,

$(w, \lambda, k_0) \vdash_M (w, a, k_1) \vdash_M (w, ab, k_2) \vdash_M (w, abc, k_3)$,
$(v, \lambda, l_0) \vdash_M (v, a, l_1) \vdash_M (v, ab, l_2) \vdash_M (v, abc, l_3)$.

Necessarily,

$(w', \lambda, 1) \vdash_M (w', a, 2) \vdash_M (w', ab, 3) \vdash_M (w', abc, 4)$,
$(v', \lambda, 1) \vdash_M (v', a, 2) \vdash_M (v', ab, 3) \vdash_M (v', abc, 4)$.

where $w' = w(k_0, 5), v' = v(l_0, 5)$ since $M_{\text{GEN}}^{\leftarrow}$ only moves forward one input letter at each step.

Let $w' = a_1a_2a_3a_4a_5, v' = a_1'a_2'a_3'a_4'a_5', a_i, a_i' \in \Sigma_{\text{RNA}}$.

Assume that $b \neq \underline{\text{s}}$. Thus, there is only one possibility for a_3; that is $a_3 = a_3'$. Assume also that $a \neq \underline{\text{s}}$, then $a_2 = a_2'$. So, assume $a = \underline{\text{s}}$. Thus, $a_1a_2, a_1'a_2'$ is equal to UC or AG. However, no two elements in $\phi_{\text{GEN}}^{-1}(d)$, for any $d \in \Gamma_{\text{AA}}$, can start with G and C. Thus, there is only one value for $a_2; a_2 = a_2'$. Then, assume $c \neq \underline{\text{s}}$. Then, $a_4 = a_4'$. Assume $c = \underline{\text{s}}$. Thus, a_3a_4 is either equal to UC or AG. But, there is only one possibility for a_3 and so there is also only one possibility for a_4. Hence, if $b \neq \underline{\text{s}}, a_2a_3a_4 = a_2'a_3'a_4'$.

Assume that $b = \underline{\text{s}}$. Then there is only one possibility for a_2 since there is no two words in $\phi^{-1}(d)$, for some $d \in \Gamma_{\text{AA}}$, that has U and A in the middle. Thus, there is also only one possibility for a_3, since $a_2 = \text{U}$ implies $a_3 = \text{C}$ and $a_2 = \text{A}$ implies $a_3 = \text{G}$. Also, there is only one possibility for a_4, since no two words of $\phi^{-1}(d)$, for some $d \in \Gamma_{\text{AA}}$ that start with either G or C can have two different middle letters. Hence, $a_2a_3a_4 = a_2'a_3'a_4'$. ■

Combining these two lemmas together, since $M_{\text{GEN}}^{\leftarrow}$ is 1-pseudo-injective and upstream, we obtain:

Proposition 0.1 *Let $x = ax'b, x \in \Gamma_{\text{AA}}^+$ with $a, b \in \Gamma$. Then*

$$(w, \lambda, 1) \vdash_{M_{\text{GEN}}^{\leftarrow}}^* (w, ax'b, k),$$
$$(v, \lambda, 1) \vdash_{M_{\text{GEN}}^{\leftarrow}}^* (v, ax'b, l),$$

implies $w(2) \cdots w(k+1) = v(2) \cdots v(l+1)$.

We are unsure if this result is known in another form, but a formal mathematical proof that it is necessarily true follows quite easily using the frameshift machine. This is interesting mathematically and also potentially interesting in the study of various genetic codes. If (unrealistically), translation were forced to proceed by shifting forward one ribonucleotide, as opposed to three, then given a string of amino acids, there would only be a single possible transcript that could code for that particular protein (except for the first and last nucleotide). This is in sharp contrast to normal translation, where a single codon can code for as many as six amino acids. This could be an important property of the standard genetic code. Note however, that it is not possible to obtain all possible sequences of amino acids using this approach. From the perspective of bioinformatics, it may be of use for studying frameshifting. For example, it may prove beneficial, given one or more proteins to "fill in" the "missing" amino acids and then uniquely determine the coding mRNA. We will explore the relevance of this idea next.

Although examining such a sequence of amino acids as above would be unusual, it would be useful to have two protein sequences which were obtained from the same RNA, but in two consecutive reading frames (likely after a +1 or -1 frameshift). Here, we will start with two such protein sequences (that are in two consecutive reading frames) and then try to determine the original RNA. This will be relevant for any $\{0,1\}$ or $\{0,-1\}$ frameshifting where there is at most one occurrence of frameshifting in the gene. There are many such biological examples of this scenario including the dnaX gene in *E. coli, V. Cholerae, Y. pestis* and *N. meningitidis* [5].

As a first step towards this problem, we start with a slightly easier problem. The next two propositions takes as "input" two protein segments produced in two consecutive reading frames, and tries to determine which reading frame each protein segment originated within. For arbitrary segments, this is not always possible. We start by developing a complete characterization of exactly which segments it is possible to do this for, and which segments for which this is ambiguous. It turns out that this characterization forms a regular language. We will take the two protein segments as input in the form $(a_1, b_1) \cdots (a_n, b_n)$, where $a_1 a_2 \cdots a_n$ is the first segment and $b_1 b_2 \cdots b_n$ is the second.

Let us first define a function ρ used in the characterization of the next proposition. For $a, b \in \Gamma_{\text{AA}} \cup \{\lambda, \$\}, a \neq \lambda$ or $b \neq \lambda$, let

$$\begin{aligned}\rho(a,b) = \quad \{cdef \mid \quad & c,d,e,f \in \Sigma_{\text{RNA}}, \text{ either } \phi_{\text{GEN}}(a) = cde \text{ or } a = \lambda \text{ or } (a = \$, cde \in E_{\text{SP}}), \\ & \text{either } \phi_{\text{GEN}}(b) = def \text{ or } b = \lambda \text{ or } (b = \$, def \in E_{\text{SP}})\}.\end{aligned}$$

If a and b are produced in two consecutive reading frames respectively, then $\rho(a,b)$ are exactly those ribonucleotides which can produce a in the first frame and b in the second. The \$ will represent the end of a protein associated with a stop codon. Let $\hat{\Gamma}_{\text{AA}} = \{(a,b) \mid a, b \in \Gamma_{\text{AA}} \cup \{\$\}, \rho(a,b) \neq \emptyset, \rho(b,a) \neq \emptyset\}$. The set $\hat{\Gamma}_{\text{AA}}$ is relevant as we are trying to understand the situations whereby it is ambiguous as to which protein segment occurs in which reading frame (whether a occur in the first frame while b occurs in the second, or vice versa). This set along with ρ at each of these values is presented in Table 2. Then we can show the following:

Proposition 0.2 *Let M be an extended frameshift machine. Let $w, v \in \Sigma^*_{\text{RNA}}$. Then*

$$(w, \lambda, i) \vdash_M (w, a, i+3), (w, \lambda, i+1) \vdash_M (w, b, i+4),$$

$$(v, \lambda, j) \vdash_M (v, b, j+3), (v, \lambda, j+1) \vdash_M (v, a, j+4)$$

if and only if $w(i,4) \in \rho(a,b)$ and $v(j,4) \in \rho(b,a)$.

$\rho(\mathrm{s},\mathrm{l}) = \{\mathrm{UCUU}, \mathrm{UCUC}, \mathrm{UCUA}, \mathrm{UCUG}\}$
$\rho(\mathrm{l},\mathrm{s}) = \{\mathrm{CUCU}, \mathrm{CUCC}, \mathrm{CUCA}, \mathrm{CUCG}\}$
$\rho(\mathrm{a},\mathrm{r}) = \{\mathrm{GCGU}, \mathrm{GCGC}, \mathrm{GCGA}, \mathrm{GCGG}\}$
$\rho(\mathrm{r},\mathrm{a}) = \{\mathrm{CGCU}, \mathrm{CGCC}, \mathrm{CGCA}, \mathrm{CGCG}\}$
$\rho(\mathrm{e},\mathrm{r}) = \{\mathrm{GAGA}, \mathrm{GAGG}\}$,
$\rho(\mathrm{r},\mathrm{e}) = \{\mathrm{AGAA}, \mathrm{AGAG}, \mathrm{CGAA}, \mathrm{CGAG}\}$
$\rho(\mathrm{i},\mathrm{y}) = \{\mathrm{AUAU}, \mathrm{AUAC}\}$,
$\rho(\mathrm{y},\mathrm{i}) = \{\mathrm{UAUU}, \mathrm{UAUC}, \mathrm{UAUA}\}$
$\rho(\mathrm{v},\mathrm{c}) = \{\mathrm{GUGU}, \mathrm{GUGC}\}$,
$\rho(\mathrm{c},\mathrm{v}) = \{\mathrm{UGUU}, \mathrm{UGUC}, \mathrm{UGUA}, \mathrm{UGUG}\}$
$\rho(\mathrm{t},\mathrm{h}) = \{\mathrm{ACAU}, \mathrm{ACAC}\}$,
$\rho(\mathrm{h},\mathrm{t}) = \{\mathrm{CACU}, \mathrm{CACC}, \mathrm{CACA}, \mathrm{CACG}\}$
$\rho(\mathrm{f},\mathrm{l}) = \{\mathrm{UUUA}, \mathrm{UUUC}\}$,
$\rho(\mathrm{l},\mathrm{f}) = \{\mathrm{CUUU}, \mathrm{CUUC}\}$
$\rho(\mathrm{f},\mathrm{f}) = \{\mathrm{UUUU}, \mathrm{UUUC}\}$,
$\rho(\mathrm{p},\mathrm{p}) = \{\mathrm{CCCU}, \mathrm{CCCC}, \mathrm{CCCA}, \mathrm{CCCG}\}$
$\rho(\mathrm{k},\mathrm{k}) = \{\mathrm{AAAA}, \mathrm{AAAG}\}$,
$\rho(\mathrm{g},\mathrm{g}) = \{\mathrm{GGGU}, \mathrm{GGGC}, \mathrm{GGGA}, \mathrm{GGGG}\}$
$\rho(\mathrm{l},\mathrm{l}) = \{\mathrm{CUUA}, \mathrm{CUUG}\}$.

Table 2: The function ρ restricted to $\hat{\Gamma}_{\mathrm{AA}}$.

Proof. Let $a, b \in \Gamma_{\mathrm{AA}}$. "$\Leftarrow$" Let $x \in \rho(a,b)$ and $y \in \rho(b,a)$. Then by inspection of the definition of ρ and similarly with y, $\phi_{\mathrm{GEN}}(x(1)x(2)x(3)) = a$ and $\phi_{\mathrm{GEN}}(x(2)x(3)x(4)) = b$. Hence, the first statement follows.

"$\Rightarrow$" This also follows by inspection. If one exhaustively tests all pairs of codons where the last two ribonucleotides of the first are the same as the first two of the second, such that both $\rho(a,b)$ and $\rho(b,a)$ are defined, we arrive at ρ restricted to $\hat{\Gamma}_{\mathrm{AA}}$. ■

We need two more definitions for the characterization. Let $c, d, e, f \in \Sigma_{\mathrm{RNA}}$. Let

$$\eta(c,d,e,f) = \{(a,b) \mid cc'd'd \in \rho(a,b), ee'f'f \in \rho(b,a), c', d', e', f' \in \Sigma_{\mathrm{RNA}}, a, b \in \Gamma_{\mathrm{AA}}\}.$$

Consider the following language:

$$\begin{aligned} L_{\mathrm{duo}} = \quad \{(a_1,b_1)\cdots(a_n,b_n) \mid \quad & \exists c_i, d_i, e_i, f_i, 1 \le i \le n, (a_i,b_i) \in \eta(c_i,d_i,e_i,f_i), \\ & d_i = c_{i+1}, f_i = e_{i+1}, 1 \le i < n\}. \end{aligned}$$

Proposition 0.3 *Let M be an extended frameshift machine. Then*

$$(w,\lambda,1) \vdash_M (w,a_1,4) \vdash_M \cdots \vdash_M (w, a_1\cdots a_n, 1+3n), \tag{1}$$
$$(w,\lambda,2) \vdash_M (w,b_1,5) \vdash_M \cdots \vdash_M (w, b_1\cdots b_n, 2+3n), \tag{2}$$
$$(v,\lambda,1) \vdash_M (v,b_1,4) \vdash_M \cdots \vdash_M (v, b_1\cdots b_n, 1+3n), \tag{3}$$
$$(v,\lambda,2) \vdash_M (v,a_1,5) \vdash_M \cdots \vdash_M (v, a_1\cdots a_n, 2+3n), \tag{4}$$

for some $w, v \in \Sigma^*_{\mathrm{RNA}}$, $a_1, \ldots, a_n, b_1, \ldots, b_n \in \Gamma_{\mathrm{AA}}$ if and only if $(a_1,b_1)\cdots(a_n,b_n) \in L_{\mathrm{duo}}$. Moreover, if the first statment is true, then for each i, $1 \le i \le n$, $w(3(i-1)+1,4) \in \rho(a_i,b_i)$ and $v(3(i-1)+1,4) \in \rho(b_i,a_i)$.

Proof. "$\Rightarrow$". Let w, v satisfy the first condition. Thus, for $1 \le i \le n, w(3(i-1)+1,4) \in \rho(a_i,b_i)$ and $v(3(i-1)+1,4) \in \rho(b_i,a_i)$ by Proposition 0.2. For each $1 \le i \le n, (a_i,b_i) \in \eta(w(3(i-1)+1), w(3(i-1)+4), v(3(i-1)+1), v(3(i-1)+4))$. Also, for each $1 \le i < n, w(3(i-1)+4) = w(3((i+1)-1)+1)$ and $v(3(i-1)+4) = v(3((i+1)-1)+1)$ and hence $(a_1,b_1)\cdots(a_n,b_n) \in L_{\mathrm{duo}}$.

"$\Leftarrow$" Let $(a_1,b_1)\cdots(a_n,b_n)$. Thus, there exist $c_i, d_i, e_i, f_i, 1 \le i \le n$, $(a_i,b_i) \in \eta(c_i,d_i,e_i,f_i)$ and for i, $1 \le i < n, d_i = c_{i+1}, f_i = e_{i+1}$.

Then, $c_ic'_id'_id_i \in \rho(a_i,b_i), e_ie'_if'_if_i \in \rho(b_i,a_i)$, for some $c'_i, d'_i, e'_i, f'_i \in \Sigma_{\mathrm{RNA}}$. Let

$$w = c_1c'_1d'_1 \overset{(=d_1)}{c_2} c'_2d'_2 \cdots \overset{(=d_{n-1})}{c_n} c'_nd'_nd_n$$

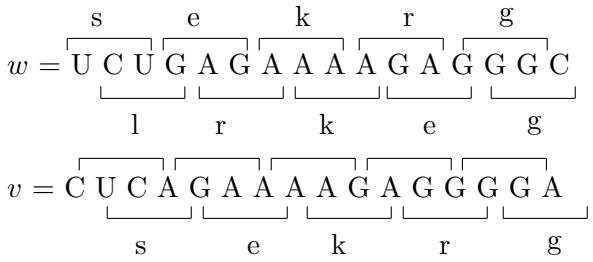

Figure 1: An example where $(\mathrm{s},\mathrm{l})(\mathrm{e},\mathrm{r})(\mathrm{k},\mathrm{k})(\mathrm{r},\mathrm{e})(\mathrm{g},\mathrm{g}) \in L_{\mathrm{duo}}$.

and

$$v = e_1 e_1' f_1' \overset{(=f_1)}{e_2} e_2' f_2' \cdots \overset{(=f_{n-1})}{e_n} e_n' f_n' f_n.$$

Then, by Proposition 0.2, for each i, $1 \leq i \leq n$,

$$(w, \lambda, 3(i-1)+1) \vdash_M (w, a_i, 3(i-1)+4), (w, \lambda, 3(i-1)+2) \vdash_M (w, b_i, 3(i-1)+5),$$

$$(v, \lambda, 3(i-1)+1) \vdash_M (v, b_i, 3(i-1)+4), (v, \lambda, 3(i-1)+2) \vdash_M (v, a_i, 3(i-1)+5).$$

Hence, the statement follows. ■

Thus, L_{duo} characterizes all pairs of protein segments which can be read in two consecutive reading frames where we cannot determine which protein comes from which reading frame (without additional knowledge, for example, the RNAs).

Example 0.1 *Consider the two protein segments* $\alpha = \mathrm{sekrg}$ *and* $\beta = \mathrm{lrkeg}$*. If these two segments are produced starting in two consecutive reading frames, then one cannot tell if* α *is in the first frame and* β *is in the second frame or* β *is in the first and* α *is in the second. Indeed, consider the diagram in Figure 0.1 which provides two substantially different RNAs, where the first has* α *in the first frame and* β *in the second frame, and the second RNA has them reversed. This follows immediately from Proposition 0.3 (without needing to construct the RNAs) by noticing that* $(\mathrm{s},\mathrm{l})(\mathrm{e},\mathrm{r})(\mathrm{k},\mathrm{k})(\mathrm{r},\mathrm{e})(\mathrm{g},\mathrm{g}) \in L_{\mathrm{duo}}$*. If we start with some string in* L_{duo}*, then we can guarantee that it can be read in opposite reading frames from (potentially different) RNAs. Moreover, because* L_{duo} *is a complete description of such strings, we cannot arrive at such an ambiguity as to which protein is in which reading frame without being in* L_{duo}*. Notice also that this language is prefix-closed and subword-closed. That means that if we have some string in this language, every prefix, and every subword of that string is also in this language.*

In each string of L_{duo}, the presence of some character in a string depends only on the previous character in the string. Consider Table 3 and its caption. Indeed, this table describes what amino acid pairs can appear directly after another amino acid pair in $\hat{\Gamma}_{\mathrm{AA}}$.

Then, we can make a deterministic finite automaton which accepts the language L_{duo}. We are able to collapse states based on similarity in terms of the symbols that can come next. This corresponds to the label in the last column of Table 3. Let $M_{\mathrm{duo}} = (Q, \hat{\Gamma}_{\mathrm{AA}}, q_0, F, \delta)$ where $Q = \{1, \ldots, 9\}, F = Q, q_0 = 1$. Each input letter can only go into one state. Let θ be a function from $\hat{\Gamma}_{\mathrm{AA}}$ into Q defined by mapping the amino acid pair in the first column of the table above to the label in the last column. Then we define the transition function δ as in Table 4. This DFA has an input alphabet of size 19, 9 states and 68 transitions and it accepts L_{duo}.

AA pairs	in	out	next	label
(s, l)	(U, C)	$\Sigma_{\text{RNA}} \times \Sigma_{\text{RNA}}$	all	1
(l, s)	(C, U)	$\Sigma_{\text{RNA}} \times \Sigma_{\text{RNA}}$	all	1
(a, r)	(G, C)	$\Sigma_{\text{RNA}} \times \Sigma_{\text{RNA}}$	all	1
(r, a)	(C, G)	$\Sigma_{\text{RNA}} \times \Sigma_{\text{RNA}}$	all	1
(e, r)	{G} × {A, C}	{A, G} × {A, G}	(r, e), (e, r), (k, k), (g, g)	2
(r, e)	{A, C} × {G}	{A, G} × {A, G}	(r, e), (e, r), (k, k), (g, g)	2
(i, y)	(A, U)	{U, C} × {U, C, A}	(s, l), (l, s), (y, i), (h, t), (f, f), (p, p), (l, l)	3
(y, i)	(U, A)	{U, C, A} × {U, C}	(s, l), (l, s), (i, y), (t, h), (f, f), (p, p), (l, l)	4
(v, c)	(G, U)	{U, C} × {U, C, A, G}	(s, l), (l, s), (r, a), (y, i), (c, v), (h, t), (f, f), (p, p), (l, l)	5
(c, v)	(U, G)	{U, C, A, G} × {U, C}	(s, l), (l, s), (a, r), (i, y), (v, c), (t, h), (f, f), (p, p), (l, l)	6
(t, h)	(A, C)	{U, C} × {U, C, A, G}	(s, l), (l, s), (r, a), (y, i), (c, v), (h, t), (f, f), (p, p), (l, l)	5
(h, t)	(C, A)	{U, C, A, G} × {U, C}	(s, l), (l, s), (a, r), (i, y), (v, c), (t, h), (f, f), (p, p), (l, l)	6
(f, l)	(U, C)	{A, C} × {U, C}	(l, s), (i, y), (t, h), (p, p), (l, l)	7
(l, f)	(C, U)	{U, C} × {A, C}	(s, l), (y, i), (h, t), (p, p), (l, l)	8
(f, f)	(U, U)	{U, C} × {U, C}	(s, l), (l, s), (f, f), (p, p), (l, l)	9
(p, p)	(C, C)	$\Sigma_{\text{RNA}} \times \Sigma_{\text{RNA}}$	all	1
(k, k)	(A, A)	{A, G} × {A, G}	(r, e), (e, r), (k, k), (g, g)	2
(g, g)	(G, G)	$\Sigma_{\text{RNA}} \times \Sigma_{\text{RNA}}$	all	1
(l, l)	(C, C)	{A, G} × {A, G}	$(r, e), (e, r), (k, k), (g, g)$	2

Table 3: In each column respectively, we describe the amino acid pair (a, b), the pair (C, D) where C are the possible nucleotides for the first letter in $\rho(a, b)$ and D are the possible letters for the first letter in $\rho(b, a)$, pair (E, F) where E are the nucleotides which can appear in the last letter in $\rho(a, b)$ and F are those that can appear in the last letter in $\rho(b, a)$, the next possible amino acid pair, and lastly a label associated with different values for the next amino acid pair (a label based on the different values for the fourth column).

$\delta(1, x) = \theta(x), \forall x \in \hat{\Gamma}_{\text{AA}}$.
$\delta(2, x) = \theta(x), \forall x \in \{(\text{r}, \text{e}), (\text{e}, \text{r}), (\text{k}, \text{k}), (\text{g}, \text{g})\}$.
$\delta(3, x) = \theta(x), \forall x \in \{(\text{s}, \text{l}), (\text{l}, \text{s}), (\text{y}, \text{i}), (\text{h}, \text{t}), (\text{p}, \text{p}), (\text{f}, \text{f}), (\text{l}, \text{l})\}$.
$\delta(4, x) = \theta(x), \forall x \in \{(\text{s}, \text{l}), (\text{l}, \text{s}), (\text{i}, \text{y}), (\text{t}, \text{h}), (\text{p}, \text{p}), (\text{f}, \text{f}), (\text{l}, \text{l})\}$.
$\delta(5, x) = \theta(x), \forall x \in \{(\text{s}, \text{l}), (\text{l}, \text{s}), (\text{y}, \text{i}), (\text{h}, \text{t}), (\text{r}, \text{a}), (\text{c}, \text{v}), (\text{p}, \text{p}), (\text{f}, \text{f}), (\text{l}, \text{l})\}$.
$\delta(6, x) = \theta(x), \forall x \in \{(\text{s}, \text{l}), (\text{l}, \text{s}), (\text{i}, \text{y}), (\text{t}, \text{h}), (\text{a}, \text{r}), (\text{v}, \text{c}), (\text{p}, \text{p}), (\text{f}, \text{f}), (\text{l}, \text{l})\}$.
$\delta(7, x) = \theta(x), \forall x \in \{(\text{l}, \text{s}), (\text{i}, \text{y}), (\text{t}, \text{h}), (\text{p}, \text{p}), (\text{l}, \text{l})\}$.
$\delta(8, x) = \theta(x), \forall x \in \{(\text{s}, \text{l}), (\text{y}, \text{i}), (\text{h}, \text{t}), (\text{p}, \text{p}), (\text{l}, \text{l})\}$.
$\delta(9, x) = \theta(x), \forall x \in \{(\text{s}, \text{l}), (\text{l}, \text{s}), (\text{p}, \text{p}), (\text{f}, \text{f}), (\text{l}, \text{l})\}$.

Table 4: The definition of the transition function in M_{duo}.

Given a string L_{duo}, $(a_1, b_1) \cdots (a_n, b_n)$, it is possible not only to determine if this string is in L_{duo}, but also to determine all possible RNAs which could have produced this pair, depending upon whether $a_1 \cdots a_n$ is in the first frame or the second.

Let w and v be as in equations (1), (2), (3), (4) of Proposition 0.3. Then, for every i, $1 \leq i \leq n, w(3(i-1)+1, 4) \in \rho(a_i, b_i), v(3(i-1)+1, 4) \in \rho(b_i, a_i)$. Given each i, and the pair (a_i, b_i), then the second and third ribonucleotides of each string in $\rho(a_i, b_i)$ and $\rho(b_i, a_i)$ can be uniquely determined (by inspection of Table 2). Moreover, the only pair for which the first nucleotide is not unique is $\rho(\text{R}, \text{E})$. Thus, if $w(3(i-1)+1, 4) \in \rho(\underline{\text{r}}, \underline{\text{e}})$ and either $i = 1$ or the nucleotide at the end of $\rho(a_{i-1}, b_{i-1})$ can end with both A and C, then the ribonucleotide at position $3(i-1)+1$ can be either A or C. This occurs if either $i = 1$ or $(a_{i-1}, b_{i-1}) \in \{(\underline{\text{s}}, \underline{\text{l}}), (\underline{\text{l}}, \underline{\text{s}}), (\underline{\text{h}}, \underline{\text{t}}), (\underline{\text{c}}, \underline{\text{v}}), (\underline{\text{y}}, \underline{\text{i}}), (\underline{\text{a}}, \underline{\text{r}}), (\underline{\text{r}}, \underline{\text{a}}), (\underline{\text{f}}, \underline{\text{l}}), (\underline{\text{p}}, \underline{\text{p}}), (\underline{\text{g}}, \underline{\text{g}})\}$. Lastly, the nucleotide at position $3(n-1)+4$ is always ambiguous since $\rho(a_n, b_n)$ necessarily has different last letters. Similarly with v. Therefore, we construct Algorithm 1 which provides a simple regular expression for all possible values of w and v of minimal length in equations (1), (2), (3), (4). Each letter of each regular expression is either a single ribonucleotide or a set of ribonucleotides in the case that there is a set of possibilities. More generally, Algorithm 1 finds the largest i such that $(a_1, b_1) \cdots (a_i, b_i) \in L_{\text{duo}}$ and determines the two regular expressions $\overline{w}$ and $\overline{v}$ for this prefix. If $i = n$ then the whole segment is in L_{duo}. Essentially, the algorithms scans each letter in M_{duo} until the prefix is not in $L(M_{\text{duo}})$. The variable q will be the current state of the automaton M_{duo}.

Example 0.2 *We will continue the example in Example 0.1. On the protein pair* $(\text{s}, \text{l})(\text{e}, \text{r})(\text{k}, \text{k})(\text{r}, \text{e})(\text{g}, \text{g})$*, the output of Algorithm 1 would be* $i = 5$ *and*

$$\overline{w} = \text{UCUGAGAAAAGAGGG}\{\text{U}, \text{C}, \text{A}, \text{G}\}, \quad \overline{v} = \text{CUC}\{\text{A}, \text{C}\}\text{GAAAAGAGGGG}\{\text{U}, \text{C}, \text{A}, \text{G}\}.$$

This expression for $\overline{w}$ *is exactly the set of possible RNAs which can produce* sekrg *in the first frame with* lrkeg *in the second. While the expression for* $\overline{v}$ *is exactly those which can be produced in switched frames.*

Of course, we are not only interested in words in L_{duo}, but also in those where it is not ambiguous as to which protein is being read in which reading frame. However, now that we have an algorithm which determines the possible RNAs, we can extend this algorithm so that it works with strings that are not in L_{duo}, and in such a case, will disambiguate as to which protein is in which reading frame. In this way, it will use Algorithm 1 first and when it reads a letter which renders the prefix out of L_{duo}, it will eliminate either $\overline{w}$ or $\overline{v}$ as a possible regular expression, leaving only the other. The algorithm will terminate when we reach a letter that is not in $\Gamma_{\text{AA}} \times \Gamma_{\text{AA}}$. In Algorithm 3 below, we will introduce the handling of stop codons and situations where one protein segment is longer than the other.

If $\alpha \in L_{\text{duo}}$, then Algorithm 1 can be used to create two regular expressions which are exactly the possible RNAs of minimal length generating those protein segments. If $\alpha \notin L_{\text{duo}}$ and there is another letter in $\Gamma_{\text{AA}} \times \Gamma_{\text{AA}}$, then Algorithm 2 will determine which one of isW or isV is true (at most one can be true, otherwise a longer prefix would be in L_{duo}), and if the former is true, will determine a regular expression $\overline{w}$ consisting of exactly the set of words w of minimal length such that the following is true:

$$\begin{aligned} &(w, \lambda, 1) \vdash (w, a_1, 4) \vdash \cdots \vdash (w, a_1 \cdots a_j, 1+3j), \\ &(w, \lambda, 2) \vdash (w, b_1, 5) \vdash \cdots \vdash (w, b_1 \cdots b_j, 2+3j), \end{aligned} \tag{5}$$

Algorithm 1: determine prefix and dual RNA regular expressions

input: $\alpha = (a_1, b_1) \cdots (a_n, b_n)$, $a_j, b_j \in \Gamma_{\mathrm{AA}} \cup \{\lambda, \$\}$,$M_{\mathrm{duo}} = (Q, \hat{\Gamma}_{\mathrm{AA}}, F, q_0, \delta)$.
output: i, the longest prefix such that $(a_1, b_1) \cdots (a_i, b_i) \in L_{\mathrm{duo}}$,
regular expressions $\overline{w}, \overline{v}$ for all possible w, v of minimal length
corresponding to $(a_1, b_1) \cdots (a_i, b_i)$ in equations (1), (2), (3), (4) of Proposition 0.3.

$i \leftarrow 1$
inDuo $\leftarrow$ true //this will be true if the prefix of length i is in L_{duo}
$q \leftarrow 1$ //start state of M_{duo}
while $i \leq n$ and inDuo = true
$q \leftarrow \delta(q, (a_i, b_i))$
if q is defined //true if $(a_1, b_1) \cdots (a_i, b_i) \in L_{\mathrm{duo}}$
if $(a_i, b_i) \leftarrow (\underline{\mathrm{r}}, \underline{\mathrm{e}})$ and ($i = 1$ or $i - 1 \in \{(\underline{\mathrm{s}}, \underline{\mathrm{l}}), (\underline{\mathrm{l}}, \underline{\mathrm{s}}), (\underline{\mathrm{h}}, \underline{\mathrm{t}}), (\underline{\mathrm{c}}, \underline{\mathrm{v}}), (\underline{\mathrm{y}}, \underline{\mathrm{i}}), (\underline{\mathrm{a}}, \underline{\mathrm{r}}),$
$(\underline{r}, \underline{a}), (\underline{f}, \underline{l}), (\underline{p}, \underline{p}), (\underline{g}, \underline{g})\}$) then $\overline{w}(3(i-1)+1) \leftarrow \{\mathrm{A}, \mathrm{C}\}$.
//as discussed above.
else let $\overline{w}(3(i-1)+1)$ be the unique first character of words in $\rho(a_i, b_i)$.
if $(a_i, b_i) = (\underline{\mathrm{e}}, \underline{\mathrm{r}})$ and ($i = 1$ or $i - 1 \in \{(\underline{\mathrm{s}}, \underline{\mathrm{l}}), (\underline{\mathrm{l}}, \underline{\mathrm{s}}), (\underline{\mathrm{t}}, \underline{\mathrm{h}}), (\underline{\mathrm{v}}, \underline{\mathrm{c}}), (\underline{\mathrm{i}}, \underline{\mathrm{y}}), (\underline{\mathrm{a}}, \underline{\mathrm{r}}),$
$(\underline{r}, \underline{a}), (\underline{l}, \underline{f}), (\underline{p}, \underline{p}), (\underline{g}, \underline{g})\}$) then $\overline{v}(3(i-1)+1) \leftarrow \{\mathrm{A}, \mathrm{C}\}$.
else let $\overline{v}(3(i-1)+1)$ be the unique first character of words in $\rho(b_i, a_i)$.
let $\overline{w}(3(i-1)+2)$ and $\overline{w}(3(i-1)+3)$ be unique 2nd, 3rd chars of words in $\rho(a_i, b_i)$.
let $\overline{v}(3(i-1)+2)$ and $\overline{v}(3(i-1)+3)$ be unique 2nd, 3rd chars of words in $\rho(b_i, a_i)$.
i++
else inDuo $\leftarrow$ false.
let $\overline{w}(3(i-1)+4)$ be the set of all last ribonucleotides of strings in $\rho(a_i, b_i)$.
let $\overline{v}(3(i-1)+4)$ be the set of all last ribonucleotides of strings in $\rho(b_i, a_i)$.
output $i - 1$,$\overline{w}$,$\overline{v}$

Algorithm 2: determine RNAs and order

input: $\alpha = (a_1, b_1) \cdots (a_n, b_n)$, $a_j, b_j \in \Gamma_{\text{AA}} \cup \{\lambda, \$\}$, $M_{\text{duo}} = (Q, \hat{\Gamma}_{\text{AA}}, F, q_0, \delta)$ be DFA above.
output: j such that $a_1, \ldots, a_j, b_1, \ldots, b_j \in \Gamma_{\text{AA}}$, isW, isV, $\overline{w}$ and $\overline{v}$,
let $i, \overline{w}$ and $\overline{v}$ be the outputs from Algorithm 1.
isW $\leftarrow$ true //true as long as $\overline{w}$ is still a possible regular expression
isV $\leftarrow$ true //true as long as $\overline{v}$ is still a possible regular expression
if $i = n$ or $(a_{i+1}, b_{i+1}) \notin \Gamma_{\text{AA}} \times \Gamma_{\text{AA}}$, then output i, isW, isV, $\overline{w}, \overline{v}$
and only one of $\overline{w}, \overline{v}$ is correct. //otherwise $(a_1, b_1) \cdots (a_j, b_j) \notin L_{\text{duo}}$
$j \leftarrow i + 1$
//determines if $\overline{w}$ or $\overline{v}$ is only possible reg. exp., and thus the reading frame of each protein.
let X be the set of characters in the last position of $\overline{w}$,
let Y be the set of characters in the last position of $\overline{v}$,
let X' be the set of first characters of words in $\rho(a_j, b_j)$
let Y' be the set of first characters of words in $\rho(b_j, a_j)$
if $X \cap X' = \emptyset$, isW $\leftarrow$ false.
if $Y \cap Y' = \emptyset$, isV $\leftarrow$ false. //one must be false otherwise $i = n$ or i not maximal
while $j \leq n$ and $a_j, b_j \in \Gamma_{\text{AA}}$
    if isW is true
        let X be the set of characters in the last position of $\overline{w}$,
        let X' be the set of first characters of words in $\rho(a_j, b_j)$
        let $\overline{w}(3(j-1)+1)$ be the set of characters $X \cap X'$
        let $\overline{w}(3(j-1)+2)$ and $\overline{w}(3(j-1)+3)$ be unique 2nd, 3rd chars of words in $\rho(a_j, b_j)$.
        let $\overline{w}(3(j-1)+4)$ be the set of all last ribonucleotides of strings in $\rho(a_j, b_j)$.
    if isV is true
        let Y be the set of characters in the last position of $\overline{v}$,
        let Y' be the set of first characters of words in $\rho(b_j, a_j)$
        let $\overline{v}(3(j-1)+1)$ be $Y \cap Y'$
        let $\overline{v}(3(j-1)+2)$ and $\overline{v}(3(j-1)+3)$ be unique 2nd, 3rd chars of words in $\rho(b_j, a_j)$.
        let $\overline{v}(3(j-1)+4)$ be the set of all last ribonucleotides of strings in $\rho(b_j, a_j)$.
    j++
if isW is true or isV is true, output $j-1$, isW, isV, $\overline{w}$ and $\overline{v}$.
else there is no RNA which can produce these two proteins.

while the latter determines a regular expression $\overline{v}$ consisting of exactly the set of words v of minimal length such that the following is true:

$$\begin{aligned}(v,\lambda,1) &\vdash (v,b_1,4) \vdash \cdots \vdash (v,b_1\cdots b_j,1+3j),\\ (v,\lambda,2) &\vdash (v,a_1,5) \vdash \cdots \vdash (v,a_1\cdots a_j,2+3j).\end{aligned} \tag{6}$$

Moveover, if isW is false, then there are no such words w, and if isV is false, then there are no such words v. Both these expressions only have a letter at each position or a set of letters (not a set of words). Indeed, characters at position $1+3i, 0 \le i \le j$ of $\overline{w}$ and $\overline{v}$ can be a set of characters, while those at $2+3i$ and $3+3i, 0 \le i < j$ cannot as the second and third letters of $\rho(a,b)$ are unique for each $a, b \in \Gamma_{\mathrm{AA}}$.

Example 0.3 *Let's say we have some string that is not in* L_{duo}*. For example, let* $\alpha = (\underline{\mathrm{s}},\underline{\mathrm{l}})(\underline{\mathrm{e}},\underline{\mathrm{r}})(\underline{\mathrm{k}},\underline{\mathrm{k}})(\underline{\mathrm{r}},\underline{\mathrm{e}})(\underline{\mathrm{c}},\underline{\mathrm{v}})$*. We already know that* $(\underline{\mathrm{s}},\underline{\mathrm{l}})(\underline{\mathrm{e}},\underline{\mathrm{r}})(\underline{\mathrm{k}},\underline{\mathrm{k}})(\underline{\mathrm{r}},\underline{\mathrm{e}}) \in \mathrm{L}_{\mathrm{duo}}$ *since it is a prefix of the example in Example 0.1. Indeed, one can accept this prefix using* M_{duo} *with the sequence of states* $1 \to 1 \to 2 \to 2 \to 2$*. But* $\alpha \notin L_{\mathrm{duo}}$ *since there is no transition from state* 2 *on* $(\underline{\mathrm{c}},\underline{\mathrm{v}})$*. On input* $(\underline{\mathrm{s}},\underline{\mathrm{l}})(\underline{\mathrm{e}},\underline{\mathrm{r}})(\underline{\mathrm{k}},\underline{\mathrm{k}})(\underline{\mathrm{r}},\underline{\mathrm{e}})$*, Algorithm 1 constructs* $\overline{w} = \mathrm{UCUGAGAAAAGA}\{\mathrm{A},\mathrm{G}\}$ *and* $\overline{v} = \mathrm{CUC}\{\mathrm{A},\mathrm{C}\}\mathrm{GAAAAGAG}\{\mathrm{A},\mathrm{G}\}$*. Considering every word in* $\rho(\underline{\mathrm{c}},\underline{\mathrm{v}})$ *starts with* U *and* w *ends with either* A *or* G*, we cannot have* $\overline{w}$ *as a regular expression. Therefore,* $\underline{\mathrm{sekrc}}$ *cannot be produced in the first frame while* $\underline{\mathrm{lrkev}}$ *produced in the second (equation (5) cannot be true for any* w*). However, there are words in* $\rho(\underline{\mathrm{v}},\underline{\mathrm{c}})$ *that start with* G*. Indeed, we can eliminate* $\overline{w}$ *and modify the regular expression* $\overline{v}$ *with* $\overline{v} = \mathrm{CUC}\{\mathrm{A},\mathrm{C}\}\mathrm{GAAAAGAGGUG}\{\mathrm{U},\mathrm{C}\}$ *are all possible RNAs which code for both protein segments (which makes equation (6) true for all* v *in the regular expression* $\overline{v}$*), but necessarily* $\underline{\mathrm{lrkev}}$ *is produced in the first frame and* $\underline{\mathrm{sekrc}}$ *in the second.*

The output of Algorithm 2 on this example is $\overline{v}$. Of interest is that even though this regular expression is of size greater than one (unlike the somewhat artificial example of Proposition 0.1), it is of very small size. Conversely, if there is not any frameshifting, given a sequence of amino acids, there is a large number of RNAs which could code for that sequence. Frameshifting drastically reduces the possible RNAs or the "degeneracy". The RNA would need to be extremely conserved throughout evolution except for exactly the positions with small sets in the regular expression above, in order to continue to generate both proteins.

We will extend this algorithm once more to take into account stop codons and reaching the ends of individual protein segments. Algorithm 2 at present takes in strings of ordered pairs between two protein segments and only continues while we have letters in $\Gamma_{\mathrm{AA}} \times \Gamma_{\mathrm{AA}}$. Of course, one protein could end with a stop codon and the other keep going or perhaps one or two of the protein segments are incomplete proteins and of different lengths. We will use the new symbol \$ to represent the end of a protein associated with a stop codon. Then α is over $(\Gamma_{\mathrm{AA}} \cup \{\lambda,\$\}) \times (\Gamma_{\mathrm{AA}} \cup \{\lambda,\$\})$ where for all j, $a_j \neq \lambda$ or $b_j \neq \lambda$, and $a_j \in \{\$,\lambda\}$ implies $a_{j+1}\cdots a_n = \lambda$, $b_j \in \{\$,\lambda\}$ implies $b_{j+1}\cdots b_n = \lambda$. We will create Algorithm 3 which determines all possible RNAs of minimal length generating the segments.

After using Algorithm 2, j will be the largest such that $(a_1,b_1)\cdots(a_j,b_j) \in \Gamma_{\mathrm{AA}} \times \Gamma_{\mathrm{AA}}$. Notice that it is impossible for any RNA to produce a protein pair α if $(\$,\$)$ is a letter, since $\rho(\$,\$)$ is empty.

We will explain case 3 of Algorithm 3 in detail with case 4 being symmetric using $\overline{v}$ instead of $\overline{w}$, and cases 1 and 2 being simpler.

In case 3, we see that the ribonucleotides which produces a_j and a_{j+1} can be any two codons producing $a_j a_{j+1}$ such that there is a stop codon starting at position two, and the first nucleotide is

Algorithm 3: determine full RNAs

input: $\alpha = (a_1, b_1) \cdots (a_n, b_n)$, $a_j, b_j \in \Gamma_{\rm AA} \cup \{\lambda, \$\}, a_j \neq \lambda$ or $b_j \neq \lambda$,
$a_j \in \{\$, \lambda\}$ implies $a_{j+1} \cdots a_n = \lambda$, $b_j = \{\$, \lambda\}$ implies $b_{j+1} \cdots b_n = \lambda$,
$M_{\rm duo} = (Q, \hat{\Gamma}_{\rm AA}, F, q_0, \delta)$ be DFA above.
output: isW, isV, $\overline{w}, \overline{v}$

let j, isW, isV, $\overline{w}, \overline{v}$ be the outputs from Algorithm 2
if $j = n$ then output isW, isV, $\overline{w}, \overline{v}$
$j++$ //then necessarily either $a_j \notin \Gamma_{\rm AA}$ or $b_j \notin \Gamma_{\rm AA}$
let X be the set of characters in the last position of $\overline{w}$,
let Y be the set of characters in the last position of $\overline{v}$,
let X' be the set of first characters of words in $\rho(a_j, b_j)$
let Y' be the set of first characters of words in $\rho(b_j, a_j)$
if $X \cap X' = \emptyset$, isW $\leftarrow$ false
if $Y \cap Y' = \emptyset$, isV $\leftarrow$ false

//case 1
if isW, and either $a_j \in \Gamma_{\rm AA} \cup \{\$\}, b_j = \lambda$ or $a_j = \$, b_j \in \Gamma_{\rm AA}$, or $a_j \in \Gamma_{\rm AA}, b_j = \$, j = n$
    if $b_j = \lambda$ let Z be those of $\phi^{-1}_{\rm GEN}(a_j)$ which start with a letter from X
    else let Z be those of $\rho(a_j, b_j)$ which start with a letter from X,
    remove last letter of $\overline{w}$ and append Z
    $j++$
//case 2
if isV, and either $b_j \in \Gamma_{\rm AA} \cup \{\$\}, a_j = \lambda$ or $b_j = \$, a_j \in \Gamma_{\rm AA}$, or $b_j \in \Gamma_{\rm AA}, a_j = \$, j = n$
    if $a_j = \lambda$ let Z be those of $\phi^{-1}_{\rm GEN}(b_j)$ which start with a letter from Y
    else let Z be those of $\rho(b_j, a_j)$ which start with a letter from Y,
    remove last letter of $\overline{w}$ and append Z
    $j++$
//case 3
if isW, and $a_j \in \Gamma_{\rm AA}, b_j = \$, j < n$ //thus $a_j \in \Gamma_{\rm AA}, a_{j+1} \in \Gamma_{\rm AA} \cup \{\$\}$,$b_{j+1} \cdots b_n = \lambda$
    let Z be words of 6 nucleotides which are in $\phi^{-1}_{\rm GEN}(a_j)\phi^{-1}_{\rm GEN}(a_{j+1})$ which start with one of the last letters of $\overline{w}$ and letters 2 through 4 is in $E_{\rm SP}$,
    remove last letter of $\overline{w}$ and append Z
        $j \leftarrow j + 2$
//case 4
if isV, and $b_j \in \Gamma_{\rm AA}, a_j = \$, j < n$ //thus $b_j \in \Gamma_{\rm AA}, b_{j+1} \in \Gamma_{\rm AA} \cup \{\$\}$,$a_{j+1} \cdots a_n = \lambda$
    let Z be words of 6 nucleotides which are in $\phi^{-1}_{\rm GEN}(b_j)\phi^{-1}_{\rm GEN}(b_{j+1})$ which start with one of the last letters of $\overline{v}$ and letters 2 through 4 is in $E_{\rm SP}$,
    remove last letter of $\overline{v}$ and append Z
        $j \leftarrow j + 2$

for $j \leftarrow j$ to n,
    if isW and $a_j \neq \lambda$, append $\phi^{-1}_{\rm GEN}(a_j)$ to $\overline{w}$
    if isW and $b_j \neq \lambda$, append $\phi^{-1}_{\rm GEN}(b_j)$ to $\overline{w}$
    if isV and $a_j \neq \lambda$, append $\phi^{-1}_{\rm GEN}(a_j)$ to $\overline{v}$
    if isV and $b_j \neq \lambda$, append $\phi^{-1}_{\rm GEN}(b_j)$ to $\overline{v}$
output isW, isV, $\overline{w}, \overline{v}$.

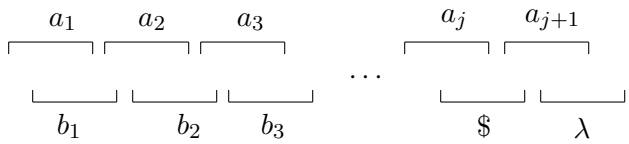

Figure 2: Case 3 of Algorithm 3.

any for which the RNA producing $(a_1, b_1) \cdots (a_{j-1}, b_{j-1})$ can end (this is the set of last characters of $\overline{w}$). Then, the for loop at the end appends any codons which can produce $a_{j+1} \cdots a_n$, as the top segment (see Figure 2) until the protein pair has been finished.

Proposition 0.4 *Let $a_1 \cdots a_n, b_1 \cdots b_m \in (\Gamma_{\text{AA}} \cup \{\$\})^*, a_1 \cdots a_{n-1}, b_1 \cdots b_{m-1} \in \Gamma_{\text{AA}}^*$. Let M be an extended frameshift machine. Then we can determine all w (if they exist) and all v (if they exist) such that the following are true:*

1. $(w, \lambda, 1) \vdash (w, a_1, 4) \vdash \cdots \vdash (w, a_1 \cdots a_j, 1 + 3j)$,
   $(w, \lambda, 2) \vdash (w, b_1, 5) \vdash \cdots \vdash (w, b_1 \cdots b_j, 2 + 3j)$, *for some* $w \in \Sigma_{\text{RNA}}^*$,

2. $(v, \lambda, 1) \vdash (v, b_1, 4) \vdash \cdots \vdash (v, b_1 \cdots b_j, 1 + 3j)$,
   $(v, \lambda, 2) \vdash (v, a_1, 5) \vdash \cdots \vdash (v, a_1 \cdots a_j, 2 + 3j)$, *for some* $v \in \Sigma_{\text{RNA}}^*$,

and produce regular expressions for each in time $O(m + n)$.

Notice that $(\$, a) \notin \hat{\Gamma}_{\text{AA}}, (a, \$) \notin \hat{\Gamma}_{\text{AA}}$ for any $a \in \Gamma_{\text{AA}} \cup \{\$\}$, which means that it is impossible for both isW and isV to be true if either there is a \$ in both $\{a_1, \ldots, a_n\}$ and $\{b_1, \ldots, b_m\}$, or if there is a letter in α which is in $\Gamma_{\text{AA}} \times \{\$\}$, or $\{\$\} \times \Gamma_{\text{AA}}$. Thus, we arrive at the following proposition:

Proposition 0.5 *Let $a_1 \cdots a_n, b_1 \cdots b_m \in (\Gamma_{\text{AA}} \cup \{\$\})^*, a_1 \cdots a_{n-1}, b_1 \cdots b_{m-1} \in \Gamma_{\text{AA}}^*$ and either $a_i = \$, b_i \in \Sigma$ or $b_i = \$, b_i \in \Sigma$, for some i. Let M be an extended frameshift machine. Then at most one of the following can be true:*

1. $(w, \lambda, 1) \vdash (w, a_1, 4) \vdash \cdots \vdash (w, a_1 \cdots a_j, 1 + 3j)$,
   $(w, \lambda, 2) \vdash (w, b_1, 5) \vdash \cdots \vdash (w, b_1 \cdots b_j, 2 + 3j)$, *for some* $w \in \Sigma_{\text{RNA}}^*$,

2. $(v, \lambda, 1) \vdash (v, b_1, 4) \vdash \cdots \vdash (v, b_1 \cdots b_j, 1 + 3j)$,
   $(v, \lambda, 2) \vdash (v, a_1, 5) \vdash \cdots \vdash (v, a_1 \cdots a_j, 2 + 3j)$, *for some* $v \in \Sigma_{\text{RNA}}^*$.

We can determine which is true, and all possible words in Σ_{RNA}^* for which this is true in time $O(m + n)$.

Indeed, Algorithm 3 above operates in linear time (in fact, it operates in a unidirectional manner on the input, scanning each protein segment one amino acid at a time). It will determine regular expressions $\overline{w}$ for all w of minimal length making Proposition 0.5 (1) true, and all $\overline{v}$ for all v of minimal length making Proposition 0.5 (2) true.

Therefore, if we have two protein segments, we can use these two propositions on the protein segments where they start to differ from each other. Then we can determine all possible RNAs which could code for them if there is one occurrence of frameshifting, and as long as the two protein segments are in two consecutive reading frames. This would be the case with either $\{0, 1\}$- or $\{0, -1\}$-frameshifting which are the most common.

General algorithms

Next, we consider the following biologically-motivated question: given a finite set of transcribed genes and a fixed translation apparatus (that is, the translation mechanisms of a single organism, with well-defined frameshifting rules), can we efficiently determine all possible translational products, allowing for frameshift events? It turns out that we are able to build a nondeterministic finite automaton (NFA) accepting all such products in time linear in the sum of the lengths of the transcribed genes and thus able to decide membership of an arbitrary protein product in quadratic time. The power of this algorithm comes from the fact that it is sufficiently general, that it will work for *any* type of frameshifting, as long as it is functional (for which all known types are). It suffices to specify the type of machine, and the algorithm is automatically provided.

Let Σ be an alphabet and $L \subseteq \Sigma^*$, L finite. Then let $s_L = \sum_{w \in L} |w|$.

Lemma 0.3 *Let $M = (\Sigma, \Gamma, \tau, E, \delta)$ be a fixed functional frameshifting machine. Let $L \subseteq \Sigma^{\geq \tau}$ with $|L| = m$. Then, there exists a nondeterministic finite automaton $M' = (Q, \Gamma, q_1, F, \delta')$ such that $|Q| = s_L - \tau m + 1, |\delta'| \leq s_L(2\tau - 1)$ and $M(L) = L(M')$ and we can construct M' in time $O(s_L)$.*

Proof. Let $L = \{w_1, \ldots, w_m\} \subseteq \Sigma^{\geq \tau}$. For $1 \leq i \leq m$ and $1 \leq j \leq |w_i| - \tau + 1$, let $b_{(i,j)} = \delta(w_i(j, \tau))$. Let $Q = \{q_1\} \cup \{q_{(i,j)} \mid 1 \leq i \leq m, 2 \leq j \leq |w_i| - \tau + 1\}$ and we create the symbols $q_{(i,1)}$ and set $q_{(i,1)} = q_1$, for every i, $1 \leq i \leq m$. We create the transitions: $q_{(i,j)} \in \delta'(q_{(i,l)}, b_{(i,l)})$ if and only if $(u, w_i(l, \tau), v, b_{(i,l)}, k) \in \delta, l + \tau + k = j, u \leq_{\mathrm{s}} w_i^{\leftarrow}(l-1), v \leq_{\mathrm{p}} w_i^{\rightarrow}(l+\tau)$. Let $F = \{q_{(i,j)} \mid w_i(j, \tau) \in E, 1 \leq i \leq m, 1 \leq j \leq |w_i|\}$.

Then $|Q| = s_L - \tau m + 1$ (there are $(|w_1| - \tau) + (|w_2| - \tau) + \cdots + (|w_m| - \tau) + 1 = s_L - \tau m + 1$ states) and $|\delta'| \leq m + s_L(2\tau - 1)$ (for each position of each word, there are at most $2\tau - 1$ subsequent positions and since M is functional, at most $2\tau - 1$ transitions. Moreover, M' can be constructed in $O(s_L)$ time since M is fixed.

It suffices to prove that $M(L) = L(M')$.

"$\subseteq$" First, if $\lambda \in M(L)$, then $\lambda \in L(M')$. Assume then, that $(w_i, \alpha_1, k_1) \vdash_M (w_i, \alpha_2, k_2) \vdash_M \cdots \vdash_M (w_i, \alpha_x, k_x)$, $1 \leq i \leq m, x > 1$, where $\alpha_1 = \lambda, k_1 = 1$ and $\alpha_j = \alpha_{j-1} b_{(i,j-1)}$, for $1 < j \leq x$.

We know $q_{(i,1)} \in \delta'(q_1, \lambda)$ ($q_1 = q_{(i,1)}$ and δ' is the transition function extended to $Q \times \Sigma^*$ as usual for finite automata [10]). Assume that $q_{(i,k_j)} \in \delta'(q_1, \alpha_j)$, for $j, 1 \leq j < x$. We also know $(w_i, \alpha_{x-1}, k_{x-1}) \vdash_M (w_i, \alpha_x, k_x)$. So, $(u, w_i(k_{x-1}, \tau), v, b_{(i,x-1)}, l) \in \delta$, where $k_{x-1} + \tau + l = k_x, u \leq_{\mathrm{s}} w_i^{\leftarrow}(k_{x-1} - 1), v \leq_{\mathrm{p}} w_i^{\rightarrow}(k_{x-1} + \tau)$. Then, $q_{(i,k_x)} \in \delta'(q_{(i,k_{x-1})}, b_{(i,k_{x-1})})$ and thus $q_{(i,k_x)} \in \delta'(q_1, \alpha_x)$.

Hence, $M(L) \subseteq L(M')$.

"$\supseteq$" If $\lambda \in L(M')$, then $\lambda \in M(L)$. Assume that, for some i, $1 \leq i \leq m$, $x \geq 2$, $q_{(i,k_{j+1})} \in \delta'(q_{(i,k_j)}, b_j)$, $b_j \in \Gamma$ for all j, $1 \leq j < x$. Then $(w_i, \lambda, 1) \vdash_M^* (w_i, \lambda, 1)$. Let $\alpha = b_{(i,1)} \cdots b_{(i,x-1)}$.

Assume that $(w_i, \lambda, 1) \vdash_M^* (w_i, \alpha^{\leftarrow}(j-1), k_j)$, where $1 \leq j < x$. But $b_j = b_{(i,k_j)} = \delta(w_i(k_j, \tau))$. We already know $q_{(i,k_x)} \in \delta'(q_{(i,k_{x-1})}, b_{x-1})$. Then, $(u, w_i(k_{x-1}, \tau), v, b_{(i,k_{x-1})}, k) \in \delta, k_{x-1} + \tau + k = k_x, u \leq_{\mathrm{s}} w_i^{\leftarrow}(k_{x-1} - 1), v \leq_{\mathrm{p}} w_i^{\rightarrow}(k_{x-1} + \tau)$. Thus, $(w_i, \lambda, 1) \vdash_M^* (w_i, \alpha^{\leftarrow}(x-1), k_x)$.

Hence, $L(M') \subseteq M(L)$. ■

It is known that, given an NFA $M = (Q, \Sigma, q_0, F, \delta)$ and $z \in \Sigma^*$, we can decide whether $z \in L(M)$ in time $O(|Q| \cdot |z|)$ [13]. Thus, we can decide whether $z \in M(L)$ in time $O(s_L \cdot |z|)$. One can also just use dynamic programming, whereby a matrix is calculated with the states on one axis and the sequence z on the other. For each column i, it will contain a 1 if each row where $z^{\leftarrow}(i)$ can be read ends up in that state, and 0 otherwise. Because each state can only have $2\tau - 1$ subsequent states, we can calculate this matrix in time $O(s_L|z|)$. If $z \in M(L)$, then $|z| \leq s_L$, so we can also decide whether $z \in M(L)$ in time $O(s_L^2)$.

Proposition 0.6 *Let $M = (\Sigma, \Gamma, \tau, E, \delta)$ be a fixed, functional frameshifting machine. Let $L \subseteq \Sigma^{\geq \tau}$ and let $z \in \Gamma^*$. Then we can decide whether $z \in M(L)$ in time $O(s_L \cdot |z|)$.*

This is quite fast and general, although it is still an open question as to whether the time complexity can be improved while maintaining the generality for "realistic" types of frameshifting much like that of the previous section.

From a biological standpoint, it may be more interesting to ask the opposite question: given a set of desired proteins and a fixed translational mechanism which allows frameshift events, can we efficiently determine a set of gene transcripts which would translate into the given proteins? The fast algorithms of the previous section were dealing with a simplified version of this question. Here we will try to solve the problem completely in a general way, but the solution does not run as fast as the algorithms in the previous section. Indeed, we show that, as above, we can construct a λ-NFA (this is a nondeterministic finite automaton with additional λ-transitions [10]) accepting this set in time linear in the sum of the inputs and decide membership in the set in quadratic time.

Lemma 0.4 *Let $M = (\Sigma, \Gamma, \tau, E, \delta)$ be a fixed, functional frameshifting machine. Let l_1 be the length of the longest left context of M and let r_1 be the maximum of the longest right context of M and $2\tau - 1$. Let $L \subseteq \Gamma^+$ with $|L| = m$. Then, there exists a λ-NFA $M' = (Q, \Sigma, q_0, F, \delta')$ such that $|Q| = (|\Sigma| + 1)^{l_1 + r_1 + \tau}(s_L + m + 1), |\delta'| \leq (|\Sigma| + 2\tau)^{l_1 + r_1 + \tau + 1}(s_L + m - 1)$ and $M^{-1}(L) = L(M')$. We can construct M' in time $O(|s_L|)$.*

Proof. Intuitively, we create a λ-NFA by keeping three buffers in the states, one to hold the left context, one to hold the current reading frame and one to hold the right context. By reading the input, we fill up the buffers using transitions of the form (7), (8), (9), (10). We also keep track of a position inside one of the words of L in the state. Then, if there is a transition of M that matches the current reading frame of the buffer, the left and right buffers, and the letter at the position of the word of L, then we shift the buffers by the frameshift amount plus τ, and increase the position inside the word of L by one. The formal proof is quite a bit more technical and is to follow.

Let $L = \{z_1, \ldots, z_m\} \subseteq \Gamma^+$. Let $X = \{(i, j) \mid 1 \leq i \leq m, 0 \leq j \leq |z_i|\} \cup \{0\}$.

Let $\Sigma_1 = \Sigma \cup \{\#\}$, $\#$ a new symbol (which we will use like a blank symbol). Let d^i be a function from Σ_1^* to Σ_1^* which eliminates the first i characters. Let $Q = \Sigma_1^{l_1} \times \Sigma_1^{\tau} \times \Sigma_1^{r_1} \times X, q_0 = (\#^{l_1}, \#^{\tau}, \#^{r_1}, 0), F = \Sigma_1^{l_1} \times E \times \Sigma_1^{r_1} \times \{(i, j) \mid 1 \leq i \leq m, j = |z_i|\}$ and δ' is defined as follows:

$$
\begin{aligned}
q &\in \delta'(q, a), a \in \Sigma, q \in F && (7)\\
(\#^{l_1}, \#^{\tau-1}a, \#^{r_1}, (i, 0)) &\in \delta'((\#^{l_1}, \#^{\tau}, \#^{r_1}, 0), a), a \in \Sigma, 1 \leq i \leq m && (8)\\
(\#^{l_1}, \#^{l}ya, \#^{r_1}, (i, 0)) &\in \delta'((\#^{l_1}, \#^{l+1}y, \#^{r_1}, (i, 0)), a), a \in \Sigma, 1 \leq i \leq m, y \in \Sigma^{\leq \tau - 1}, l < \tau && (9)\\
(\#^{j_1}u, y, va\#^{j_2-1}, (i, j)) &\in \delta'((\#^{j_1}u, y, v\#^{j_2}, (i, j)), a), (i, j) \in X, a \in \Sigma, 0 \leq j_1 \leq l_1, j_2 > 0 && (10)\\
&\quad u \in \Sigma^{\leq l_1}, v \in \Sigma^{\leq r_1 - 1}, y \in \Sigma^{\tau}, (\#^{j_1}u, y, v\#^{j_2}, (i, j)) \in Q - F
\end{aligned}
$$

and

$$((d^{k+\tau}(\#^{j_1}uyv))^{\leftarrow}(l_1), (d^{k+\tau}(yv))^{\leftarrow}(\tau), (d^{k+\tau}(v\#^{r_1}))^{\leftarrow}(r_1), (i, j+1)) \in \delta'((\#^{j_1}u, y, v\#^{j_2}, (i, j)), \lambda),$$

$j_1, j_2 \geq 0, y \in \Sigma^{\tau}, u \in \Sigma^{\leq l_1}, v \in \Sigma^{\leq r_1}, a \in \Sigma, (i, j), (i, j+1) \in X, (\#^{j_1}u, y, v\#^{j_2}, (i, j)) \in Q - F$, if and only if $(u', y, v', z_i(j+1), k) \in \delta, u' \leq_{\mathrm{s}} u, v' \leq_{\mathrm{p}} v$.

"$\subseteq$" Let $w \in M^{-1}(L)$. Necessarily, $|w| \geq \tau$. Then $(w, \alpha_1, k_1) \vdash_M \cdots \vdash_M (w, \alpha_l, k_l)$, where $\alpha_j \in \Gamma^*, k_j \in \mathbb{N}(|w|)$ for $1 \leq j \leq l, k_1 = 1, \alpha_1 = \lambda, \alpha_l = z_{i'}$, for some i', $1 \leq i' \leq m$ and $w(k_l, \tau) \in E$. Let p is the smallest integer such that $w(k_p, \tau) \in E$ and $\alpha_p = z_i$, for some i, $1 \leq i \leq m$, which must exist. Assume $p = 1$. Then $w(1, \tau) \in E$ and it is clear that $(\#^{l_1}, w(1, \tau), \#^{r_1}, (i, 0)) \in$

$\delta'(q_0, w(1,\tau)) \cap F \cap \delta'(q_0, w)$. Assume $p > 1$. Then, $|w| > \tau$ and $l > 1$. We will prove by induction that for every j, $1 \leq j \leq p$, $(u, w(k_j, \tau), v, (i, |\alpha_j|)) \in \delta'(q_0, w^{\leftarrow}(k))$, where $|\alpha_j| = j - 1, u = \#^{l_1 - k_j + 1} w(1) \cdots w(k_j - 1)$, if $k_j \leq l_1$ and $u = w(k_j - l_1, l_1)$, otherwise (if we have not read enough characters to have the left buffer full, then it is preceeded by the appropriate number of # symbols), and $v = w(k_j + \tau) \cdots w(|w|)\#^{r_1 - |w| + k_j + \tau - 1}$ with $k = |w|$ if $|w| - k_j - \tau + 1 \leq r_1$ and $v = w(k_j + \tau, r_1)$ with $k = k_j + \tau + r_1 - 1$, otherwise (the right buffer is full if there is enough characters of w left, otherwise it is padded with # symbols).

For the base case, $(\#^{l_1}, w(1,\tau), v, (i, 0)) \in \delta'(q_0, w^{\leftarrow}(k))$ where $v = w(\tau + 1) \cdots w(|w|)\#^{r_1 - |w| + \tau}$ with $k = |w|$ if $|w| - \tau \leq r_1$ and $v = w(\tau + 1, r_1)$ with $k = \tau + r_1$ otherwise, by using one transition from (8), $\tau - 1$ transitions from (9) and at most $|v|$ transitions from (10).

Assume that, for $1 \leq j < p$, $(u, w(k_j, \tau), v, (i, |\alpha_j|)) \in \delta'(q_0, w^{\leftarrow}(k))$, where $|\alpha_j| = j - 1, u = \#^{l_1 - k_j + 1} w(1) \cdots w(k_j - 1)$, if $k_j \leq l_1$ and $u = w(k_j - l_1, l_1)$, otherwise and $v = w(k_j + \tau) \cdots w(|w|)\#^{r_1 - |w| + k_j + \tau - 1}$ with $k = |w|$ if $|w| - k_j - \tau + 1 \leq r_1$ and $v = w(k_j + \tau, r_1)$ with $k = k_j + \tau + r_1 - 1$, otherwise. We know that $(w, \alpha_j, k_j) \vdash_M (w, \alpha_{j+1}, k_{j+1})$ and so $(u', w(k_j, \tau), v', b, m') \in \delta, u' \leq_{\mathrm{s}} w^{\leftarrow}(k_j - 1), v' \leq_{\mathrm{p}} w^{\rightarrow}(k_j + \tau), \alpha_{j+1} = \alpha_j b, b \in \Gamma, k_{j+1} = k_j + m' + \tau$. Then, $|\alpha_{j+1}| = j, u' \leq_{\mathrm{s}} u, v' \leq_{\mathrm{p}} v$ (since u and v are of the longest possible lengths). Thus, if $u'' = (d^{m' + \tau}(uw(k_j, \tau)v)^{\leftarrow}(l_1), y'' = (d^{m' + \tau}(w(k_j, \tau)v))^{\leftarrow}(\tau), v'' = (d^{m' + \tau}(v\#^{r_1}))^{\leftarrow}(r_1)$, then

$$(u'', y'', v'', (i, j)) \in \delta'((u, w(k_j, \tau), v, (i, j - 1)), \lambda),$$

and so $(u'', y'', v'', (i, j)) \in \delta'(q_0, w^{\leftarrow}(k))$. Here $y'' = w(k_j + m' + \tau, \tau) = w(k_{j+1}, \tau), u'' = \#^{l_1 - k_{j+1} + 1} w(1) \cdots w(k_{j+1} - 1)$ if $k_{j+1} \leq l_1$ and $u'' = w(k_{j+1} - l_1, l_1)$ otherwise. Then, if $k = |w|$, then $v'' = w(k_{j+1} + \tau) \cdots w(|w|)\#^{r_1 - |w| + k_{j+1} + \tau - 1}, k_{j+1} \leq r_1$, and we are done. Assume $k < |w|$. In this case, the right buffer is not full and we will fill it up as much as possible. Then $(u'', y'', v''', (i, j)) \in \delta'(q_0, w^{\leftarrow}(\min\{|w|, k_{j+1} + \tau + r_1 - 1\})$ where $v'' = x\#^l, x \in \Sigma^*, l > 1, v''' = xx'\#^{l'}, xx' = w(k_{j+1} + \tau) \cdots w(\min\{|w|, k_{j+1} + \tau + r_1 - 1\})$ using transitions of type (10), and we are done, by induction. Hence, $w \in L(M')$.

"$\supseteq$" Let $w \in L(M')$. Thus, $q_{j+1} \in \delta'(q_j, a_{j+1}), 0 \leq j < l', a_{j+1} \in \Sigma \cup \{\lambda\}, a_1 \cdots a_{l'} = w$ with $q_{l'} \in F$. Let p be the smallest number such that $q_p \in F$. Let $1 \leq i_1 < \cdots < i_k = p$ be those indices such that $a_{i_j} = \lambda$. If $k = 0$, then $w(1,\tau) \in E$ and $w \in M^{-1}(L)$. Assume $k > 0$. Then, there exists i, $1 \leq i \leq m$ such that, for each j, $1 \leq j \leq k$, $q_{i_j - 1} = (\#^{l_1 - |u_j|} u_j, y_j, v_j \#^{r_1 - |v_j|}, (i, j - 1))$, for some $u_j, y_j, v_j \in \Sigma^*$,

$$\begin{aligned} q_{i_j} &= (\#^{l_1 - |u_j''|} u_j'', y_j'', v_j'' \#^{r_1 - |v_j''|}, (i, j)) \\ &= ((d^{m_j + \tau}(\#^{l_1 - |u_j|} u_j y_j v_j))^{\leftarrow}(l_1), (d^{m_j + \tau}(y_j v_j)^{\leftarrow}(\tau)), (d^{m_j + \tau}(v_j \#^{r_1})^{\leftarrow}(r_1), (i, j)) \end{aligned}$$

and so $(u_j', y_j, v_j', z_i(j), m_j) \in \delta, u_j' \leq_{\mathrm{s}} u_j, v_j' \leq_{\mathrm{p}} v_j$. We wish to prove by induction that $(w, \alpha_1, x_1) \vdash_M \cdots \vdash_M (w, \alpha_k, x_k)$, where for every j, $1 \leq j < k$, $x_1 = 1, x_{j+1} = x_j + m_j + \tau, \alpha_1 = \lambda, \alpha_{j+1} = \alpha_j z_i(j)$ and $w(x_j, \tau) = y_j, u_j \leq_{\mathrm{s}} w^{\leftarrow}(x_j - 1), v_j \leq_{\mathrm{p}} w^{\rightarrow}(x_j + \tau)$ and $|a_1 \cdots a_{i_j}| = x_j + \tau + |v_j| - 1$ (the last condition is necessary for the inductive step in order to prove that $v_l \leq_{\mathrm{p}} w^{\rightarrow}(x_l + \tau)$).

For the base case, one can see by the construction that $w(1,\tau) = y_1, u_1 \leq_{\mathrm{s}} w^{\leftarrow}(0) = \lambda, v_1 \leq_{\mathrm{p}} w^{\rightarrow}(\tau + 1)$ and $|a_1 \cdots a_{i_1}| = \tau + |v_1|$.

Assume that $(w, \alpha_1, x_1) \vdash_M \cdots \vdash_M (w, \alpha_l, x_l)$, $1 \leq l < k$, where, for every j, $1 \leq j < l$, $x_1 = 1, x_{j+1} = x_j + m_j + \tau, \alpha_1 = \lambda, \alpha_{j+1} = \alpha_j z_i(j)$ and $w(x_j, \tau) = y_j, u_j \leq_{\mathrm{s}} w^{\leftarrow}(x_j - 1), v_j \leq_{\mathrm{p}} w^{\rightarrow}(x_j + \tau)$ and $|a_1 \cdots a_{i_j}| = x_j + \tau + |v_j| - 1$. Notice $(u_l', y_l, v_l', z_i(l), m_l) \in \delta, u_l' \leq_{\mathrm{s}} u_l$ and $v_l' \leq_{\mathrm{p}} v_l$. We will first try to show $|a_1 \cdots a_{i_l}| = x_l + \tau + |v_l| - 1$. Then, $x_l = x_{l-1} + m_{l-1} + \tau$, by the induction hypothesis and $|v_l| = |v_{l-1}| - m_{l-1} - \tau + |a_{i_{l-1}+1} \cdots a_{i_l}|$ since the right buffer gets shifted by $m_{l-1} + \tau$ and new

characters $a_{i_{l-1}+1} \cdots a_{i_l}$ are read between $q_{i_{l-1}-1}$ and q_{i_l-1}. Thus,

$$\begin{aligned} |a_1 \cdots a_{i_l}| &= |a_1 \cdots a_{i_{l-1}}| + |a_{i_{l-1}+1} \cdots a_{i_l}| = x_{l-1} + \tau + |v_{l-1}| - 1 + |a_{i_{l-1}+1} \cdots a_{i_l}| \\ &= x_l - m_{l-1} + |v_{l-1}| - 1 + |a_{i_{l-1}+1} \cdots a_{i_l}| = x_l + |v_l| - 1 + \tau. \end{aligned}$$

Next, we will show $u_l \leq_{\mathrm{s}} w^{\leftarrow}(x_l - 1), y_l = w(x_l, \tau)$ and $v_l \leq_{\mathrm{p}} w^{\rightarrow}(x_l + \tau)$. We know $u_{l-1} \leq_{\mathrm{s}} w^{\leftarrow}(x_{l-1} - 1), y_{l-1} = w(x_j, \tau), v_{l-1} \leq_{\mathrm{p}} w^{\rightarrow}(x_{l-1} + \tau)$ by the induction hypothesis and hence after shifting by $m_{l-1} + \tau$, we determine that $u''_{l-1} \leq_{\mathrm{s}} w^{\leftarrow}(x_{l-1} + m_{l-1} + \tau - 1) = w^{\leftarrow}(x_l - 1), y''_{l-1} = y_l = w(x_{l-1} + m_{l-1} + \tau, \tau) = w(x_l, \tau), v''_{l-1} \leq_{\mathrm{p}} w^{\rightarrow}(x_{l-1} + m_{l-1} + \tau) = w^{\rightarrow}(x_l + \tau)$. Moreover, $u''_{l-1} = u_l$ and $v_l = v''_{l-1} a_{i_{l-1}+1} \cdots a_{i_l}$, as the states change between $q_{i_{l-1}}$ and q_{i_l-1} ($a_{i_l} = \lambda$ which is why v_l can end with a_{i_l}) by keeping the same left buffer and reading $a_{i_{l-1}+1} \cdots a_{i_l}$ into the right buffer. Thus $u_l = u''_{l-1} \leq_{\mathrm{s}} w^{\leftarrow}(x_l - 1)$. Indeed, we already showed $|a_1 \cdots a_{i_l}| = x_l + |v_l| - 1 + \tau$ and so the number of characters read from w, $|a_1 \cdots a_{i_l}|$ minus $|v_l| + 1$ gives the starting position of v_l in w. This is $x_l + \tau$ and hence $v_l \leq_{\mathrm{p}} w^{\rightarrow}(x_l + \tau)$. So, $u'_l \leq_{\mathrm{s}} w^{\leftarrow}(x_l - 1), v'_l \leq_{\mathrm{p}} w^{\rightarrow}(x_l + \tau)$. Thus, $(w, \alpha_l, x_l) \vdash_M (w, \alpha_{l+1}, x_{l+1})$ with $x_{l+1} = x_l + m_l + \tau$. Hence $M^{-1}(L) = L(M')$.

Also, the size of Q has the stated number of elements, and the number of transitions is less than or equal to $m|\Sigma|(|\Sigma|+1)^{l_1+r_1+\tau}) + m|\Sigma| + m(|\Sigma|+1)^{\tau-1} + (s_L + m + 1)(|\Sigma|+1)^{l_1+r_1+\tau-1} + (s_L + m + 1)(|\Sigma|+1)^{l_1+r_1+\tau}(2\tau - 1) \leq (|\Sigma| + 2\tau)^{l_1+r_1+\tau+1}(s_L + m - 1)$ ■

Similarly to Proposition 0.6, we get the following:

Proposition 0.7 *Let $M = (\Sigma, \Gamma, \tau, E, \delta)$ be a fixed frameshifting machine. Let $L \subseteq \Gamma^+$ and let $w \in \Sigma^*$. Then, we can decide whether $w \in M^{-1}(L)$ in time $O(s_L \cdot |w|)$.*

Proof. We can either build a single λ-NFA for all $z \in L$, or build separate λ-NFAs for each $z \in L$ accepting $M^{-1}(z)$ and then test if $w \in M^{-1}(z)$ for each z. We will adopt the latter approach. To test if w is in the λ-NFAs, we could compute the λ-closure and then use an NFA membership algorithm, but we will instead describe a dynamic programming approach. We will compute a dynamic programming matrix with each λ-NFA, with the states on one axis and the sequence w on the other. In the matrix, we will put a 1 at a position in column i if we can reach the row-indexed state after reading the input $w^{\leftarrow}(i)$, and a 0 otherwise. For each state, there at most $2\tau - 1$ subsequent states on λ and at most one on each letter. Thus, we can calculate the matrix on that word in time $O(|z| \cdot |w|)$ for each word $z \in L$, and then with all the λ-NFAs in time $O(s_L \cdot |w|)$. ■

Here again, this algorithm will work for any type of frameshifting, so long as the machine is functional. We next present the last algorithm.

Let $M_i = (Q_i, \Sigma, q_{0,i}, F_i, \delta_i)$ be λ-nondeterministic finite automata, for $1 \leq i \leq m$. The product automaton of $M_1, M_2, \ldots, M_m$, denoted by $M_1 \times \cdots \times M_m$, is defined as being the λ-nondeterministic finite automaton $M_1 \times \cdots \times M_m = (Q_1 \times \cdots \times Q_m, \Sigma, q_{0,1} \times \cdots \times q_{0,m}, F_1 \times \cdots \times F_m, \delta)$, where $(q_1, \ldots, q_m) \in \delta((p_1, \ldots, p_m), a)$ if and only if $q_i \in \delta_i(p_i, a)$, for all i, $1 \leq i \leq m$. Let M be a finite automaton. Then, $\overline{M}$ is the finite automaton obtained by removing loops. If we construct a finite automaton M' from a frameshift machine M as in Lemma 0.4, then $\overline{M'}$ will be acyclic, by the construction. Moreover, the only loops are using transitions of the form $q \in \delta(q, a), q \in F$. Thus, a shortest path from the initial state to a final state in $M_1 \times \cdots \times M_m$ will be present in $\overline{M_1 \times \cdots \times M_m}$.

We now consider a more restricted version of the above question: given a fixed translational mechanism and a finite set of protein products, does there exist a single gene transcript which can generate exactly that set of proteins?

Proposition 0.8 *Let $L = \{z_1, \ldots, z_m\} \subseteq \Gamma^+$ and let $M = (\Sigma, \Gamma, \tau, E, \delta)$ be a fixed, functional, frameshift machine. Then, there is an algorithm to determine whether there exists a string $w \in \Sigma^{\geq \tau}$ such that $L \subseteq M(w)$, and moreover, if such a string exists, a representative of smallest length can be constructed in time $O(|z_1| \cdot \cdots \cdot |z_m|)$.*

Proof. Let $Z_i = \{z_i\}$, for $1 \leq i \leq m$ and let $M_i = (Q_i, \Sigma, q_{0,i}, F_i, \delta_i)$ be the λ-nondeterministic finite automaton constructed from Lemma 0.4 such that $M_i(Z_i) = M^{-1}(Z_i)$, for each i, $1 \leq i \leq m$. Let $M' = M_1 \times \cdots \times M_m$ be the product automaton of M_1 through M_m. Thus, $w \in L(M_1 \times \cdots \times M_m)$ if and only if $w \in L(M_1) \cap \cdots \cap L(M_m) = M^{-1}(z_1) \cap \cdots \cap M^{-1}(z_m)$. Then, there exists such a w if and only if there exists a w such that $L \subseteq M(w)$. Moreover, by the observations preceding the Lemma, it suffices to decide whether there exists $w \in L(\overline{M_1 \times \cdots \times M_m})$. Moreover, since the automaton is merely a weighted, acyclic graph with weights of 0 for a λ-edge and 1 otherwise, we can determine the length of a shortest path from an initial state (or vertex) to a final state (or vertex) in time $O(|Q'| + |\delta'|)$, where Q' is the set of states (or vertices) in M' and δ' is the set of transitions (or edges) in M' (see the standard algorithm to determine the single-source shortest paths in directed acyclic graphs, for example in section 24.2 of [14]). Indeed, $|Q'| = O(|z_1| \cdot \cdots \cdot |z_m|)$ and similarly, $|\delta'| = O(|z_1| \cdot \cdots \cdot |z_m|)$. In addition, it is possible to determine such a path in this time, so we can determine a shortest such w. Hence, we can determine whether there exists w such that $L \subseteq M(w)$ and output a smallest such w, if it exists in time $O(|z_1| \cdot \cdots \cdot |z_m|)$. ■

Conclusions

We have introduced the *frameshift machine* as a formal model of the biological event of the reading-frame shift during translation. Our model allows us to mathematically study the relationships between products and operands in both the context of an abstract transformation on strings and, more significantly, in the special case of biological systems with RNA and amino acids.

We have demonstrated that this model yields efficient methods for investigating the relationships between protein products and gene transcripts under translation with frameshifting. Specifically, we can determine the set of all potential proteins for a given set of gene transcripts and given a set of target proteins, we can efficiently determine a set of gene transcripts which will translate to the desired protein. These algorithms will work for any type of frameshifting. Moreover, if we are primarily interested in known types of frameshifting, then in this case we can determine all possible RNAs which could code for two proteins at their point of divergence via frameshifting in linear time.

In light of recent discoveries indicating that the number of genes in higher organisms is many fewer than previously hypothesized, biologists are increasingly accepting the critical role that genetic information processing events play in proteomic diversity. Frameshifting during translation is one such process. We have presented here a mathematical foundation for studying the possibilities introduced by these events and which will hopefully lead to a better understanding of this process both theoretically and *in vivo*.

Acknowledgements

This research was supported by grants from the Natural Sciences and Engineering Research Council of Canada, institutional grants of the University of Saskatchewan and the University of Western Ontario and the SHARCNET Research Chairs Program.

References

[1] Prescott, D.: Genome gymnastics: Unique modes of DNA evolution and processing in ciliates. Nature Reviews Genetics **1** (2000) 191–198

[2] Horton, T., Landweber, L.: Rewriting the information in DNA: RNA editing in kinetoplastids and myxomycetes. Microbiology **5**(6) (2002) 620–626

[3] Alberts, B.: Molecular Biology of the Cell, 5th edition. Garland Science, NY (2007)

[4] Shehu-Xhilaga, M., Crowe, S., Mak, J.: Maintenance of the gag/gag-pol ratio is important for human immunodeficiency virus type 1 RNA dimerization and viral infectivity. Journal of Virology **75** (2001) 1834–1841

[5] Baranov, P.V., Gesteland, R.F., Atkins, J.F.: Recoding: Translational bifurcations in gene expression. Gene **286** (2002) 187–201

[6] Stahl, G., McCarty, G.P., Farabaugh, P.J.: Ribosome structure: Revisiting the connection between translational accuracy and unconventional decoding. Trends in Biochemical Sciences **27**(4) (2002) 178–183

[7] Huang, X., Zhang, J.: Methods for comparing a DNA sequence with a protein sequence. Computer Applications in the Biosciences **12**(6) (1996) 497–506

[8] Guan, X., Uberbacher, E.C.: Alignments of DNA and protein sequences containing frameshift errors. Comput. Appl. Biosci. **12**(1) (February 1996) 31–40

[9] Birney, E., Thompson, J., Gibson, T.: PairWise and SearchWise: finding the optimal alignment in a simultaneous comparison of a protein profile against all DNA translation frames. Nucleic Acids Res. **24** (1996) 2730–2739

[10] Hopcroft, J., Ullman, J.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading, MA (1979)

[11] Lind, D., Marcus, B.: An introduction to symbolic dynamics and coding. Cambridge University Press, Cambridge, NY (1995)

[12] Lozupone, C., Knight, R., Landweber, L.: The molecular basis of nuclear genetic code changes in ciliates. Current Biology **11** (2001) 65–74

[13] Holub, J., Melichar, B.: Implementation of nondeterministic finite automata for approximate pattern matching. In Champarnaud, J.M., Maurel, D., Ziadi, D., eds.: WIA '98, Lecture Notes in Computer Science. Volume 1660. Springer-Verlag (1999) 92–99

[14] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, Second Edition. The MIT Press (September 2001)