Tensor Product Representations of Regular Transductions

Zhouyi Sun[1]   **Jonathan Rawski**[1,2]

[1]Massachusetts Institute of Technology   [2]San Jose State University

{szy, rawski}@mit.edu

Abstract

This paper provides a vector space characterization of regular transductions. We use finite model theory to characterize objects like strings and trees as relational structures and origin graphs to characterize input-output relations generated by transducer. We show detailed processes of using multilinear maps as function application for evaluation to compile regular transductions characterized by MSO definable origin graphs into a tensor embedding.

1 Introduction

The mathematical theory of automata provides a way to explicitly tie the complexity of linguistic patterns to specific claims about memory organization and thus provides an direct way of measuring the cognitive demands of language. Transducers, i.e. automata that produce outputs beyond "yes" or "no", have been around since the beginnings of automata theory, and have a long history in linguistics and NLP for modeling the complexity of various linguistic processes (Mohri, 1997; Heinz, 2018; Roark and Sproat, 2007).

One particular class of interest is the regular transductions, which generalize the class of regular languages. Regular languages are one of the most well-studied objects in computer science, characterized by regular expressions, finite-state automata, and statements in Monadic Second-Order logic, among others (Thomas, 1997). Linguistically, the regular class has been shown to sufficiently characterize phonological and morphological phenomena (Kaplan and Kay, 1994; Rawski et al., 2023; Dolatian et al., 2021).

The regular transductions have become far better understood in recent years. Engelfriet and Hoogeboom (2001) showed that MSO-transducers, a logical model of transducers studied in the general context of graph transductions (Courcelle, 1994; Courcelle and Engelfriet, 2012), exactly characterize the transductions realized by two-way transduc-

ers. A model of one-way transducers with registers, called streaming string transducers, has also been shown to capture the same class of transductions, which were then called regular functions (Alur and Černý, 2010).

This paper considers logical characterizations of regular functions over structures that are defined using finite model theory. Model theory has been used for comparisons of particular grammatical theories in phonology and syntax (Rogers, 1998; Pullum, 2007; Graf, 2010), and for studying the nature of linguistic structures and processes themselves (Heinz, 2018; Payne et al., 2016). Linguistic structures like strings and trees are modeled using relational information which holds among the elements characterizing a particular structure.

It is of interest to see how these models may be characterized in vector spaces. Vector space approaches to language and symbolic cognition in general have become increasingly popular during the last two decades. There is work dealing with conceptual spaces for sensory representations (Gardenfors, 2004), multilinear representations for compositional semantics (Blutner, 2009; Aerts, 2009), and dynamical systems for modeling language processes (Beim Graben et al., 2008; Tabor, 2009).

One particularly significant contribution in this area is Tensor Product Representation (Smolensky, 1990). Here, subsymbolic dynamics of neural activation patterns in a vector space description become interpreted as symbolic cognitive computations at a higher-level description by means of "filler/role" bindings via tensor products. These tensor product representations form the symbolic foundation of Harmonic Grammar and Optimality Theory, and have been successfully employed for phonological and syntactic computations (Smolensky and Legendre, 2006).

Tensor methods and (sub)regular grammars/automata have been used to evaluate and interpret neural networks (Rabanser et al., 2017).

McCoy et al. (2018) showed that recurrent neural networks (RNNs) implicitly encode tensor product representations, and Strobl et al. (2023) survey work using regular languages to test transformer language models. Nelson et al. (2020) used regular string transductions to test the generalization capacity of RNNs, finding that they failed to successfully learn them unless explicitly given machinery which enabled them to approximate the underlying two-way finite-state transducer.

There has also work on embedding logical calculi using tensors. Grefenstette (2013) introduces tensor-based predicate calculus that realizes logical operations. Yang et al. (2014) introduce a method of mining Horn clauses from relational facts represented in a vector space. Serafini and Garcez (2016) introduce logic tensor networks that integrate logical deductive reasoning and data-driven relational learning. Sato (2017) formalizes Tarskian semantics of first-order logic in vector spaces. Rawski (2019) employed Sato's method to translate model-theoretic representations and languages definable in first-order logic (the star-free and locally threshold testable sets) into tensors. This paper extends that work to consider transductions.

2   Transductions as origin graphs

Model theory, combined with logic, provides a powerful way to study and understood mathematical objects with structures (Enderton, 2001). This paper only considers finite relational models (Libkin, 2004).

Definition 1. A model signature *is a tuple* $S = \langle D; R_1, R_2, \ldots, R_m \rangle$ *where the domain* D *is a finite set, and each* R_i *is a* n_i-*ary relation over the domain.*

In this paper, the relations are at most binary.

Definition 2. A model for a set of objects M is a total, one-to-one function from M to structures whose type is given by a model signature S.

The flexibility gven by model-theoretic representations allows them to consider many things as objects, including relations between inputs and outputs. Bojańczyk (2014), attempting to solve the problem of how to decide whether two transducers generate the same input-output pairs, created a novel way to describe transductions as model-theoretic structures by using an *origin mapping*. Informally, an origin mapping, which is a total function from output positions to input positions showing which input position(s) are used for a given

output symbol. Bojańczyk et al. (2017) directly considered this relation between inputs and outputs as a structure, called an *origin graph*, for which they defined the corresponding idea of an *origin transduction*.

Definition 3. *(Bojańczyk et al., 2017) An origin string-to-string transduction (origin transduction for short) consists of an input alphabet* Σ*, an output alphabet* Γ*, and a set of origin graphs over these alphabets specifying the input position used to produce each output position.*

An origin graph with input w *and output* v *consists of:*

- *The domain which is the disjoint union of positions in* w *and positions in* v;

- *Two binary predicates for the successor relations in* w *and* v;

- *a binary predicate, the origin mapping, which is a total function from output position to input positions;*

- *a unary predicate for each* $a \in \Sigma \cup \Gamma$ *which identifies positions with label* a.
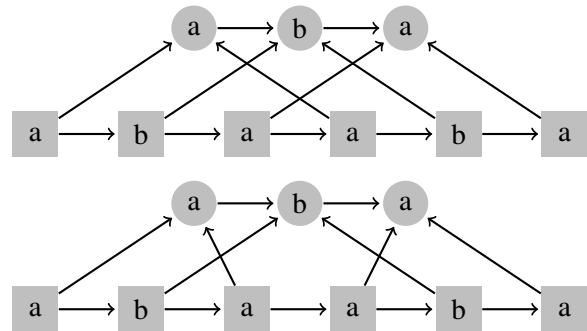

Figure 1: Visualizations of two string-to-string transductions differing only on the origin mapping.

Consider two transductions from the input string aba to the output string $abaaba$ visualized in fig. 1. In addition to the pair of input and output string ($aba \rightarrow abaaba$) involved in the transduction, the origin input position of each output position differentiates origin transductions, especially in terms of their recognizability by automata (see e.g. Dolatian et al., 2021). In fig. 1, input positions are denoted by circles. Output positions are denoted by squares. And the arrows from an output position to an input position represents the origin information of the output position. The upper figure can be seen

as the result of reduplication computed by a two-way automaton which goes back-and-forth over the input.

Origin graphs present an intriguing way to take concepts used to define classes of formal languages, and apply them to transductions by considering them as structures. This has a history in linguistics, namely through evaluations of phonological ideas. Various recent works have used formalisms resembling origin semantics to discuss Correspondence Theory (Payne et al., 2016) and rewrite-rule interaction (Meinhardt et al., 2024).

3 Logical Languages and Transduction Classes

Usually a model signature provides the vocabulary for some logical language $\mathscr{L}$, which contains N constants $\{e_1, \ldots, e_N\}$. Following notation of Sato (2017), a model $M = (D, I)$ is thus a pair of domain, a nonempty set D and an interpretation I that maps constants e_i to elements (entities, individuals) $I(e_i) \in D$ and k-ary predicate symbols r to k-ary relations $I(r) \subseteq D^k$.

An assignment a is a mapping from variables x to an element $a(x) \in D$. It provides a way of evaluating formulas containing free variables. Syntactically terms mean variables and/or constants and atomic formulas or atoms $r(t_1, \ldots, t_k)$ are comprised of a k-ary predicate symbol r and k terms $t_1, \ldots, t_k$ some of which may be variables. Formulas F in $\mathscr{L}$ are inductively constructed as usual from atoms using logical connectives (negation $\neg$, conjunction $\wedge$, disjunction $\vee$) and quantifiers ($\exists, \forall$). First-order formulas allow only quantification over elements. Monadic Second-Order (MSO) formulas additionally allow countably many second-order set variables $X, Y, \ldots$ with $x \in X$, which can be quantified over $\forall X, \exists X$. In this case, a is a mapping from set variables variables X to a set of elements $a(X) \in D$.

Sentences in this logical language define sets of strings/trees as follows. The language of a formula F is all and only those graphs whose models satisfy F. For any formula F, $\llbracket F \rrbracket_{I,a} \in \{1, 0\}$ and when $\llbracket F \rrbracket_{I,a} = 1$, we write $M \vDash_a F$ to mean the model satisfies F. However when F is closed, since $\llbracket F \rrbracket_{I,a}$ does not depend on the assignment a, we just write $\llbracket F \rrbracket$ and $M \vDash F$ if F is true in M.

There are several well-known connections between logical statements and languages classes. Most famous is Büchi (1960)'s result that lan-

guages characterizable by finite-state machines, the regular languages, are equivalent to statements in Monadic Second-Order Logic over the precedence model for strings (and successor, since precedence is MSO-definable from successor).

Courcelle (1994) lifted the idea of MSO to transductions, creating the MSO-definable analog of the regular languages. Engelfriet and Hoogeboom (2001) showed an equivalence between MSO-transducers and two-way transducers (where the read head can move back and forth on the input). Later, Alur and Černý (2010) showed another equivalence with streaming string transducers (where the two-way read head is replaced with a finite number of registers), giving the following result:

Theorem 1 (Engelfriet and Hoogeboom, 2001; Alur and Černý, 2010; Courcelle, 1994). *A transduction is regular iff it is realized by a 2-way Deterinistic Finite-state Transducer or an MSO-transducer or a Streaming String Transducer.*

This convergence of results led to particular problems of deciding whether a given transducer is equivalent to another one, or whether two transducers compute the same string relation in the same way. This is analogous to the concept of weak versus strong capacity in linguistics (Miller, 1999). Bojańczyk et al. (2017), using the origin graphs defined earlier, showed another characterization:

Theorem 2. *(Bojańczyk et al., 2017) Let G be an origin transduction, i.e., an input alphabet, an output alphabet, and a set of origin graphs over these alphabets. G is a regular function (recognized by a streaming string transducer) iff:*
bounded origin: *there is some $m \in \mathbb{N}$ such that in every origin graph from G, every input position is the origin of at most m output positions;*
k-crossing: *in every origin graph from $\mathcal{G}$, every input position is crossed by at most k output positions (An output position j crosses an input position i if the origin of j is no greater than i, and either j is the final output position, or the successor of j has its origin greater than i.);*
MSO-definable: *there is an MSO formula which is true in exactly the origin graphs from $\mathcal{G}$.*

The main goal of this paper, extending Rawski (2019), is to characterize the origin graphs with these properties via tensor calculus in order to embed transductions in vector spaces.

4 Tensor Representations of Logical Constraints

We first want to show how to embed a model domain and signature into a vector space, using tensors to encode relational information. We then show how the ingredients of a logical language (specifically, First-Order and Monadic Second-Order logics) translate to operations over tensors.

Scalars are denoted with lower case letters like a. Vectors mean column vectors and we denote them by boldface lower case letters like $\mathbf{a}$ and $\mathbf{a}$'s components by a_i. $\mathscr{D}' = \{\mathbf{e}_1, \ldots, \mathbf{e}_N\}$ is the standard basis of N-dimensional Euclidean space $\mathbb{R}^N$ where $\mathbf{e}_i = (0 \cdots, 1, \cdots, 0)^T$ is a vector that has one at the i-th position and zeros elsewhere. Such vectors are called one-hot vectors. For set variables are denoted

$\mathbf{1}$ is a vector of all ones. We assume square matrices, written by boldface upper case letters like $\mathbf{A}$. $\mathbf{I}$ is an identity matrix, and $\mathbb{1}$ is a matrix of all ones. Order-p tensors $\mathcal{A} \in \mathbb{R}^{D^p}$, are also denoted by $\{a_{i_1,\ldots,i_p}\}\,(1 \le i_1, \ldots, i_p \le N)$. $\mathcal{A}$'s component $a_{i_1,\ldots,i_p}$ is also written as $(\mathcal{A})_{i_1,\ldots,i_p}$. $(\mathbf{a} \bullet \mathbf{b}) = \mathbf{a}^T \mathbf{b}$ is the inner product of $\mathbf{a}$ and $\mathbf{b}$ whereas $\mathbf{a} \circ \mathbf{b} = \mathbf{a}\mathbf{b}^T$ is their outer product. $\mathbf{1} \circ \cdots \circ \mathbf{1}$ is a k-order tensor, and $\mathbf{1} \circ \cdots \circ \mathbf{1}\,(\mathbf{e}_{i_1}, \ldots, \mathbf{e}_{i_k}) = (\mathbf{1} \bullet \mathbf{e}_{i_1}) \cdots (\mathbf{1} \bullet \mathbf{e}_{i_k}) = 1$. Scalars, vectors, and matrices are tensors of order 0, 1, and 2 respectively.

There exists an isomorphism between tensors and multilinear maps (Bourbaki, 1989), such that any curried multilinear map

$$f : V_1 \to \ldots \to V_j \to V_k$$

can be represented as a tensor $\mathcal{T}_f \in V_k \otimes V_j \otimes \ldots \otimes V_1$. This means that tensor contraction acts as function application. This isomorphism guarantees that there exists such a tensor $\mathcal{T}^f$ for every f, such that for any $v_1 \in V_1, \ldots, v_j \in V_j$:

$$f\mathbf{v}_1 \ldots \mathbf{v}_j = \mathbf{v}_k = \mathcal{T}^f \times \mathbf{v}_1 \times \ldots \times \mathbf{v}_j \quad (1)$$

Following Sato (2017), we first isomorphically map a model M to a model M' in $\mathbb{R}^N$. We map entities $e_i \in D$ to one-hot vectors $\{\mathbf{e}_i\}$. So D is mapped to $D' = \{\mathbf{e}_1, \ldots, \mathbf{e}_N\}$, the basis of $\mathbb{R}^N$. We next map a k-ary relation r in M to a k-ary relation r' over D' which is computed by an order-k tensor $\mathcal{R} = \{r_{i_1,\ldots,i_k}\}$, whose truth value $[\![r(e_{i_1}, \ldots, e_{i_k})]\!]$ in M is given by

$$[\![r(e_{i_1}, \ldots, e_{i_k})]\!]$$

$$= \mathcal{R}(\mathbf{e}_{i_1}, \ldots, \mathbf{e}_{i_k})$$
$$= \mathcal{R} \times_{11} \mathbf{e}_{i_1} \times_{12} \cdots \times_{1,i_k} \mathbf{e}_{i_k}$$
$$= r_{i_1,\ldots,i_k} \in \{1, 0\}\,(\forall i_1, \ldots, i_k \in \{1, \ldots, N\}) \tag{2}$$

We identity r' with $\mathcal{R}$ so that $\mathcal{R}$ encodes the M-relation r. Let M' be a model (D', I') in $\mathbb{R}^N$ such that I' interprets entities by $I'(e_i) = \mathbf{e}_i (1 \le i \le N)$ and relations r by $I'(r) = \mathcal{R}$.

For the purposes of this paper, we restrict ourselves to binary relations and predicates. When r is a binary predicate, the corresponding tensor $\mathcal{R}$ is a bilinear map and represented by an adjacency matrix $\mathbf{R}$ as follows:

$$[\![(e_i, e_j)]\!] = (\mathbf{e}_i \cdot \mathbf{R}\mathbf{e}_j) = \mathbf{e}_i^T \mathbf{R}\mathbf{e}_j = r_{ij} \in \{1, 0\} \tag{3}$$

Note that when $r(x, y)$ is encoded by $\mathcal{R}$ as $(\mathbf{x} \bullet \mathbf{R}\mathbf{y})$, $r(y, x)$ is encoded by $\mathbf{R}^T$, since $(\mathbf{y} \bullet \mathbf{R}\mathbf{x}) = (\mathbf{x} \bullet \mathbf{R}^T \mathbf{y})$ holds

We next inductively define the evaluation $[\![F]\!]_{I',a'}$ of a formula F in M. Let a be an assignment in M and a' the corresponding assignment in M', so $a(x) = e_i$ iff $a'(x) = \mathbf{e}_i$. For a ground atom $r(e_{i_1}, ..., e_{i_k})$, define

$$[\![r(e_{i_1}, \ldots, e_{i_k})]\!]' = \underline{\mathbf{R}}(\mathbf{e}_{i_1}, \ldots, \mathbf{e}_{i_k})$$
$$(\forall i_1, \ldots, i_k \in \{1, \ldots, N\}). \tag{4}$$

where $\mathcal{R} = \{r_{i_1,\ldots,i_1}\}$ is a tensor encoding the M-relation r in M. By definition $[\![F]\!]_{I,a} = [\![F]\!]_{I,a}$ holds for any atom F. Negative literals are evaluated using $\neg\mathcal{R}$ defined as

$$[\![\neg r(e_{i_1}, \ldots, e_{i_k})]\!]' = \neg\mathcal{R}(\mathbf{e}_{i_1}, \ldots, \mathbf{e}_{i_k})$$
$$\text{where } \neg\mathcal{R} \overset{\text{def}}{=} \overbrace{\mathbf{1} \circ \cdots \circ \mathbf{1}}^{k} - \mathcal{R} \tag{5}$$

$\neg\mathcal{R}$ encodes an M-relation $\neg r_1$. Negation other than negative literals, conjunction, disjunction, and quantifiers are evaluated in M' as follows.

$$[\![\neg F]\!]' = 1 - [\![F]\!]' \tag{6}$$
$$[\![F_1 \wedge \cdots \wedge F_h]\!]' = [\![F_1]\!]' \cdots [\![F_h]\!]' \tag{7}$$
$$[\![F_1 \vee \cdots \vee F_h]\!]' = \min_1([\![F_1]\!]' + \ldots + [\![F_h]\!]') \tag{8}$$

$$[\![\exists y F]\!]' = \min_1(\sum_{i=1}^{N}[\![F_{y \leftarrow e_i}]\!]') \tag{9}$$

Here the operation $\min_1(x) = \min(x, 1) = x$ if $x < 1$, otherwise 1, as componentwise application. $F_{y \leftarrow e_i}$ means replacing every free occurrence

of y in F with e_i. Universal quantification over individual elements is treated as $\forall x F = \neg \exists x \neg F$.

Monadic second order logic allows variables over sets of elements in addition to first order formulas discussed above. A set variable X consisting of k entities $e_{i_1}, e_{i_2}, \ldots, e_{i_k}$ can be represented as the sum of the corresponding one-hot vectors, $\mathbf{e}_X = \sum_{e_i \in X} \mathbf{e}_i$ which is a k-hot vector. Notice here the subscript is a set of numbers X instead of a number indicating the position, e.g. i. The evaluation of ground atoms like $r(E_{i_1}, \ldots, E_{i_k})$ can stay the same, which is also true of other first order evaluations. The existential quantification of set variables can be evaluated in M' as follows.

$$\llbracket \exists X F \rrbracket' = \min_1 \left(\sum_{I \subseteq D} \llbracket F_{X \leftarrow I} \rrbracket' \right) \qquad (10)$$

Similarly, universal quantification over sets can be treated as $\forall X F = \neg \exists X \neg F$.

We can now define the properties over origin graphs using this formulation as in Theorem 2. For an origin graph, the set of its input positions is N and the set of its output positions is M. Binary relation R_{origin} defines the origin information between N and M. $R_{\text{origin}}(i, j) = 1$ when the output position j has the input position i as its origin.

For every input position x, the condition of k-crossing is equivalent to

$$\sum_{i=1}^{|M|} R_{\text{origin}}(x, i) \leq k \qquad (11)$$

The condition of bounded origin is equivalent to

$$\sum_{i=1}^{|M|} \text{cross}(x, i) \leq k \qquad (12)$$

Equating input positions withe natural numbers 1 to $|N|$ and output positions with natural numbers 1 to $|M|$, the function *cross* (see theorem 2 for definition) can be defined as

$$\text{cross}(x, i) = \begin{cases} 1 & \exists k (k \leq x \cap R(k, i)) \cap \\ & \cap (i = |M| \cup \exists l (x < l \cap \\ & \cap R(l, i+1))) \\ 0 & \text{otherwise} \end{cases}$$

Intuitively, $\text{cross}(x, i)$ returns 1 if an output position i has origin at some input position k preceding an input position x, such that i is either the last output position or its successor has origin to the right of x. It returns 0 otherwise.

5 Examples

This section presents detailed processes of compiling MSO definable origin graphs into a tensor embedding, based on Sato (2017); Rawski (2019). MSO formulas are first converted into prenex normal form. Then the formula is translated into its corresponding tensor representation by applying evaluations rules discussed in section 4. For readability, we often collapse multiple sequential $\exists$ quantifiers into one.

5.1 -t insertion

We begin with a simple process of concatenating a symbol *-t* onto the end of an output word, akin to suffixation or epenthesis. This process of *-t* insertion maps, for example, the input string *ba* to *bat*. The process can be captured by an MSO formula (in fact, a first order formula):

$$F_{-t} = \forall x (R_{\text{input}}(x) \rightarrow$$
$$\exists y (R_{\text{origin}}(x, y) \land R_{\text{equal}}(x, y) \land ((R_{\text{last-i}}(x) \land$$
$$\exists z (R_{\text{succ-o}}(y, z) \land R_{\text{origin}}(x, z) \land R_{\text{t-o}}(z) \land R_{\text{last-o}}(z)))$$
$$\lor \exists x', y' (R_{\text{succ-i}}(x, x') \land R_{\text{succ-o}}(y, y') \land$$
$$\land R_{\text{origin}}(x', y')))))) \qquad (13)$$

We assert that for any input position x ($R_{\text{input}}(x) = 1$), there exists an output position y whose origin is x ($R_{\text{origin}}(x, y) = 1$) and the labels of x and y are the same ($R_{\text{equal}}(x, y) = 1$). We follow Bojańczyk et al. (2017) in distinguishing the set of input alphabet and the set of output alphabet. For example, $R_{\text{t-output}}(z)$ checks whether position z is an output position and whether its label is *t*. In this way, unary predicates R_{input}, R_{output} and the binary predicate R_{equal} can all be defined with no difficulty.

Additionally, if x is the last input position ($R_{\text{last-i}}(x) = 1$), then y has a successor z. The origin of z is x ($R_{\text{origin}}(x, z) = 1$). Its label is *t* in the output alphabet ($R_{\text{t-output}}(z) = 1$). And it is the last position in the output ($R_{\text{last-o}}(z) = 1$). Otherwise x has its successor x' ($R_{\text{succ-i}}(x, x') = 1$) and y also has its successor y' ($R_{\text{succ-o}}(y, y') = 1$), whose origin is x' ($R_{\text{origin}}(x', y') = 1$).

The adjacency matrix defined by the binary relation R_{origin} in this case is almost an identity matrix. Suppose the input has a length of n. Then for any i less that n, (i, i) is 1. $(n, n+1)$ is also 1. All other entries are 0. It satisfies the constraints of **bounded origin** and **k-crossing** naturally, as each

input position is the origin of at most 2 output positions and is crossed by at most 2 output positions. And thus the origin transduction of *-t* insertion is recognizable by a streaming string transducer with k registers.
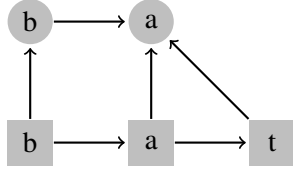

Figure 2: Visualizations of the origin graph of suffixiating *-t* to *ba*.

Because the formula of *-t* or any suffix insertion is properly first order (in fact it's subsequential (Mohri, 1997)), its origin graph has a straightforward embedding into tensor operations following results from Rawski (2019) (see also section 4).

First, the formula can be converted into prenex normal form:

$$\forall x \exists y \exists z \exists x' \exists y' (\neg R_{\text{input}}(x) \vee (R_{\text{origin}}(x,y) \wedge \\ \wedge R_{\text{equal}}(x,y) \wedge ((R_{\text{last-i}}(x) \wedge \\ \wedge (R_{\text{succ-o}}(y,z) \wedge R_{\text{origin}}(x,z) \wedge R_{\text{t-o}}(z) \wedge R_{\text{last-o}}(z)))) \vee \\ \vee (R_{\text{succ-i}}(x,x') \wedge R_{\text{succ-o}}(y,y') \wedge R_{\text{origin}}(x',y')))))) \tag{14}$$

Compiling the prenex formal formula into tensor notation, we get

$$T_{-t} = 1 - \min_1 \sum_{x=1}^{N} (1 - \min_1 \sum_{y,z,x',y'=1}^{N} \\ (\min_1 ((1 - \mathcal{R}^{\text{input}} \mathbf{e}_x) + (\mathbf{e}_x^T \mathcal{R}^{\text{origin}} \mathbf{e}_y) \bullet (\mathbf{e}_x^T \mathcal{R}^{\text{equal}} \mathbf{e}_y) \\ \bullet \min_1 ((\mathcal{R}^{\text{last-i}} \mathbf{e}_x) \bullet (\mathbf{e}_y^T \mathcal{R}^{\text{succ-o}} \mathbf{e}_z) \\ \bullet (\mathbf{e}_x^T \mathcal{R}^{\text{origin}} \mathbf{e}_z) \bullet (\mathcal{R}^{\text{t-i}} \mathbf{e}_z) \bullet (\mathcal{R}^{\text{last-o}} \mathbf{e}_z) + \\ ((\mathbf{e}_x^T \mathcal{R}^{\text{succ-i}} \mathbf{e}_{x'}) \bullet (\mathbf{e}_y^T \mathcal{R}^{\text{succ-o}} \mathbf{e}_{y'}) \bullet (\mathbf{e}_{x'}^T \mathcal{R}^{\text{origin}} \mathbf{e}_{y'} \\ )))))) \tag{15}$$

Here we can see how each of the ingredients of the logical formula maps, straightforwardly, to ingredients of the tensor formulation. Note that $\sum_{y,z,x',y'=1}^{N}$ collapses the four existential quantifiers, for ease of readability.

5.2 Copying

Next we demonstrate an MSO formula for the process of copying the input word, which is of more linguistic significance. Copying, known as reduplication in linguistics, is a common morphological process which is often argued to be among the most complex phenomena in linguistics. Copying is properly a regular function, and is one of the standard characteristic functions used to define the properties of the class, namely the linear growth property (see Rawski et al. (2023) for details).

$$F_{\text{copying}} = \forall x (R_{\text{input}}(x) \rightarrow \\ \rightarrow \exists Y, Z (R_{\text{output-path}}(Y,Z) \wedge \\ \wedge \exists y, z (R_{\text{origin}}(x,y) \wedge R_{\text{origin}}(x,z) \wedge y \in Y \wedge z \in Z \wedge \\ \wedge (\neg R_{\text{last-input}}(x) \rightarrow \\ \exists x', y', z' (R_{\text{succ-input}}(x,x') \wedge R_{\text{succ-output}}(y,y') \wedge \\ \wedge R_{\text{succ-output}}(z,z') \wedge R_{\text{origin}}(x',y') \wedge R_{\text{origin}}(x',z') \\ ))))) \tag{16}$$

The binary predicate $R_{\text{output-path}}(Y,Z)$ holds when Y and Z partition the output positions, in which Y and Z are two paths and the first position of Z is the successor of the last position of Y. It can be formalized by the conjunction of three predicates: 1) Y and Z together cover all output positions, with each position belonging exclusively to either Y or Z; 2) Both Y and Z must be paths. A path is defined as a connected graph where every position, except one, has a successor within the path, and every position, except one, is a successor of another position in the path; 3) the tail of Y has the head of X as its successor in X (the tail of a path can be defined as the only position without a successor in the path; the head of a path is not a successor of any position in the path).

The formula thus states that for every input position x, there are two output positions $y \in Y$ and $z \in Z$, whose origin is x. The part requires they bear the same label is omitted for the ease of understanding. Whenever x is not the last position in the input, y and z are also not last positions in Y and Z. Their successors y' and z' both have the successor of x, x' as their origin.

In this way, each input position is mapped to exactly two output positions and **bounded origin** is satisfied. The last input position is only crossed once by the last output position. Every other input position is crossed twice by the two output positions it mapped to. And therefore **k-crossing** is satisfied as well. Importantly, the number of copies must be linearly bounded (i.e. some pre-specified number of copies of an input string) in order to remain MSO-definable. Unbounded copying vio-

lates both constraints and is not MSO definable, nor is bounding the number of copies to some higher order, say polynomially. ([Rawski et al., 2023](#)).
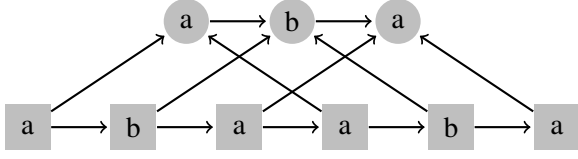

Figure 3: Visualizations of the origin graph of copying *aba*

Converting the formula into prenex normal form:

$$F_{\text{copying}} = \forall x \exists Y \exists Z \exists y \exists z \exists x' \exists y' \exists z' (\neg R_{\text{INPUT}}(x)$$
$$\vee (R_{\text{output-path}}(Y, Z)$$
$$\wedge (R_{\text{origin}}(x, y) \wedge R_{\text{origin}}(x, z) \wedge y \in Y \wedge z \in Z$$
$$\wedge (R_{\text{last-input}}(x) \vee (R_{\text{succ-input}}(x, x') \wedge R_{\text{succ-output}}(y, y')$$
$$\wedge R_{\text{succ-output}}(z, z') \wedge R_{\text{origin}}(x', y') \wedge R_{\text{origin}}(x', z')$$
$$)))))) \quad (17)$$

The formula can be compiled into tensor notation as follows:

$$\mathcal{T}_{\text{copying}} =$$
$$1 - \min_1 (\sum_{x=1}^{N} (1 - \min_1 \sum_{Y,Z \subseteq D} (\min_1 (\sum_{y,z,x',y',z'=1}^{N}$$
$$\min_1 ((1 - \mathcal{R}^{\text{origin}} \mathbf{e}_{y'}) + (\mathcal{R}^{\text{succ-output}} \times \mathbf{e}_Y \times \mathbf{e}_Z) \bullet$$
$$(\mathbf{e}_x^T \mathcal{R}^{\text{origin}} \mathbf{e}_y) \bullet (\mathbf{e}_x^T \mathcal{R}^{\text{origin}} \mathbf{e}_z) \bullet (\mathbf{e}_y \bullet \mathbf{e}_Y) \bullet (\mathbf{e}_y \bullet \mathbf{e}_Y) \bullet$$
$$\min_1 (\mathcal{R}^{\text{last-input}} \mathbf{e}_x + (\mathbf{e}_x^T \mathcal{R}^{\text{succ-input}} \mathbf{e}_{x'}) \bullet$$
$$(\mathbf{e}_y^T \mathcal{R}^{\text{succ-output}} \mathbf{e}_{y'}) \bullet (\mathbf{e}_z^T \mathcal{R}^{\text{succ-output}} \mathbf{e}_{z'}) \bullet$$
$$(\mathbf{e}_{x'}^T \mathcal{R}^{\text{origin}} \mathbf{e}_{y'}) \bullet (\mathbf{e}_{x'}^T \mathcal{R}^{\text{origin}} \mathbf{e}_{z'}))))))) \quad (18)$$

5.3 First-Last to Even-Odd mapping

In this subsection we show that in contrast to the previous example, the origin graph of First-Last to Even-Odd mapping does not satisfy the condition of **k-crossing**. Patterns of this type are unattested linguistically, though they are reminiscent of certain types of spreading patterns.

Consider the origin transduction exemplified in fig. [4](#). Every odd position in the output has the last input position as its origin. Every even output position has the first input position as its origin. The length of the output can be arbitrary. All input positions are then sandwiched between the origins of each neighboring even-odd output position pair.
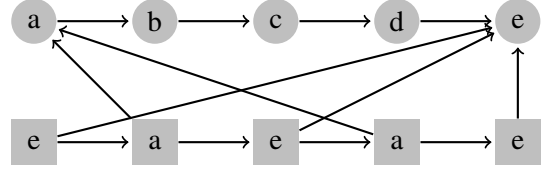

Figure 4: Visualizations of the origin graph of reversing *abcde*

And thus these input positions are crossed by each even position in the output. Suppose the length of the output is n. The crossing number is $\lfloor n/2 \rfloor$, which grows with n and is unbounded.

The essential property of origin graphs of first-last to even-odd mapping can be defined by the following MSO formula:

$$F_{\text{FLEO}} =$$
$$\exists Y_{\text{output-o}}, Y_{\text{output-e}} (R_{\text{output-o/e}}(Y_{\text{output-o}}, Y_{\text{output-e}})$$
$$\wedge \forall y (y \in Y_{\text{output-o}} \rightarrow \exists x_f (R_{\text{first-input}}(x_f)$$
$$\wedge R_{\text{origin}}(x_f, y)) \wedge (y \in Y_{\text{output-e}} \rightarrow$$
$$\exists x_l (R_{\text{last-input}}(x_l) \wedge R_{\text{origin}}(x_l, y))))) \quad (19)$$

The MSO definable binary predicate $R_{\text{output-o/e}}(Y_{\text{output-o}}, Y_{\text{output-e}})$ is true when $Y_{\text{output-o}}$ and $Y_{\text{output-e}}$ constitute a partition of the set of output positions and $Y_{\text{output-o}}$ is the set of all odd output positions while $Y_{\text{output-e}}$ denotes the set of all even output positions (see e.g. [Filiot, 2015](#)). The formula asserts that every even output position has the last input position as its origin while every odd output position has the first input position as its origin position. The part requires they bear the same label is omitted for the ease of understanding.

We can convert this formula into prenex normal form as follows:

$$\exists Y_{\text{output-o}} \exists Y_{\text{output-e}} \forall y \exists x_f \exists x_l ($$
$$R_{\text{output-o/e}}(Y_{\text{output-o}}, Y_{\text{output-e}}) \wedge$$
$$\wedge (\neg y \in Y_{\text{output-o}} \vee (R_{\text{first-input}}(x_f) \wedge R_{\text{origin}}(x_f, y)) \wedge$$
$$\wedge (\neg y \in Y_{\text{output-e}} \vee (R_{\text{last-input}}(x_l) \wedge R_{\text{origin}}(x_l, y)))))$$
$$(20)$$

We can compile this into a tensor formula as follows:

$$\mathcal{T}_{\text{FLEO}} = \min_1 \sum_{Y_{\text{output-o}}, Y_{\text{output-e}} \subseteq D} (1 - \min_1 \sum_{y=1}^{N} (1-$$

$$- \min_1 \sum_{x_f, x_l = 1}^{N} (\mathcal{R}^{\text{output-o/e}} \times \mathbf{e}_{Y_{\text{output-o}}} \times \mathbf{e}_{Y_{\text{output-e}}}) \bullet$$

$$\bullet \min_1 ((1 - \mathbf{e}_y \bullet \mathbf{e}_{Y_{\text{output-o}}}) + (\mathcal{R}^{\text{first-input}} \mathbf{e}_{x_f}) \bullet$$

$$\bullet (\mathbf{e}_{x_f}^T \mathcal{R}^{\text{origin}} \mathbf{e}_y)) \bullet \min_1 ((1 - \mathbf{e}_y \bullet \mathbf{e}_{Y_{\text{output-e}}}) +$$

$$+ (\mathcal{R}^{\text{last-input}} \mathbf{e}_{x_l}) \bullet (\mathbf{e}_{x_l}^T \mathcal{R}^{\text{origin}} \mathbf{e}_y)))) \quad (21)$$

6 Conclusion

This paper showed how to embed transductions in a vector space via operations over tensors. In particular, by using the idea of origin graphs, which represent input-output relations computed by some transducer, we embedded these graphs into tensors via finite model theory, and introduced Monadic Second-Order logical operations to compile the connectives and quantifiers. We showed how a class of origin graphs with these properties characterizing the regular transductions fits this exactly, and gave several examples motivated from linguistics.

There are several further directions this work could take. The most obvious is to consider the class of First-Order transductions on its own term. First-Order functions generalize the star-free languages (definable in first-order logic) to transductions, and correspond to restricting the underlying automaton of the transducer to be aperiodic (see Filiot et al. (2019)).

Transductions have also been extended to other structures besides strings, such as trees, which are relevant data structures in syntactic and semantic phenomena. The concept of origin information can be extended from string transducers to tree transducers, by considering the input and output graphs as tree structures ordered by dominance (Filiot et al., 2018; Winter, 2021). Therefore, tree transductions can be embedded into vector space using the same methods.

In general, the flexibility given by model theory, as well as the precision given by classes of transductions, allows for multiple characterizations of structures of interest to linguistics, computer science, and cognitive science.

References

Diederik Aerts. 2009. Quantum structure in cognition. *Journal of Mathematical Psychology*, 53(5):314–348.

Rajeev Alur and Pavol Černý. 2010. Expressiveness of streaming string transducers. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, Leibniz International Proceedings in Informatics, pages 1–12, Germany. Dagstuhl Publishing.

Peter Beim Graben, Dimitris Pinotsis, Douglas Saddy, and Roland Potthast. 2008. Language processing with dynamic fields. *Cognitive Neurodynamics*, 2(2):79–88.

Reinhard Blutner. 2009. Concepts and bounded rationality: An application of niestegge's approach to conditional quantum probabilities. In *AIP Conference Proceedings*, volume 1101, pages 302–310. AIP.

Mikołaj Bojańczyk. 2014. Transducers with origin information. In *Automata, Languages, and Programming: 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II 41*, pages 26–37. Springer.

Mikołaj Bojańczyk, Laure Daviaud, Bruno Guillon, and Vincent Penelle. 2017. Which classes of origin graphs are generated by transducers? In *ICALP 2017*.

Nicolas Bourbaki. 1989. *Commutative Algebra: Chapters 1-7*. Springer-Verlag.

J. Richard Büchi. 1960. Weak second-order arithmetic and finite automata. *Mathematical Logic Quarterly*, 6(1-6):66–92.

Bruno Courcelle. 1994. Monadic second-order definable graph transductions: a survey. *Theoretical Computer Science*, 126(1):53–75.

Bruno Courcelle and Joost Engelfriet. 2012. *Graph structure and monadic second-order logic: a language-theoretic approach*, volume 138. Cambridge University Press.

Hossep Dolatian, Jonathan Rawski, and Jeffrey Heinz. 2021. Strong generative capacity of morphological processes. In *Proceedings of the Society for Computation in Linguistics 2021*, pages 228–243.

Herbert B. Enderton. 2001. *A Mathematical Introduction to Logic*, 2nd edition. Academic Press.

Joost Engelfriet and Hendrik Jan Hoogeboom. 2001. MSO definable string transductions and two-way finite-state transducers. *ACM Transactions on Computational Logic (TOCL)*, 2(2):216–254.

Emmanuel Filiot. 2015. Logic-automata connections for transformations. In *Indian Conference on Logic and Its Applications*, pages 30–57. Springer.

Emmanuel Filiot, Olivier Gauwin, and Nathan Lhote. 2019. Logical and algebraic characterizations of rational transductions. *Logical methods in computer science*, 15.

Emmanuel Filiot, Sebastian Maneth, Pierre-Alain Reynier, and Jean-Marc Talbot. 2018. Decision problems of tree transducers with origin. *Information and Computation*, 261:311–335.

Peter Gardenfors. 2004. Conceptual spaces as a framework for knowledge representation. *Mind and Matter*, 2(2):9–27.

Thomas Graf. 2010. Logics of phonological reasoning. Master's thesis, University of California, Los Angeles.

Edward Grefenstette. 2013. Towards a formal distributional semantics: Simulating logical calculi with tensors. In *Proceedings of the Second Joint Conference on Lexical and Computational Semantics*.

Jeffrey Heinz. 2018. The computational nature of phonological generalizations. In Larry Hyman and Frans Plank, editors, *Phonological Typology*, Phonetics and Phonology, chapter 5, pages 126–195. De Gruyter Mouton.

Ronald M Kaplan and Martin Kay. 1994. Regular models of phonological rule systems. *Computational linguistics*, 20(3):331–378.

Leonid Libkin. 2004. *Elements of Finite Model Theory*. Springer.

R Thomas McCoy, Tal Linzen, Ewan Dunbar, and Paul Smolensky. 2018. Rnns implicitly implement tensor product representations. *arXiv preprint arXiv:1812.08718*.

Eric Meinhardt, Anna Mai, Eric Baković, and Adam McCollum. 2024. Weak determinism and the computational consequences of interaction. *Natural Language & Linguistic Theory*, pages 1–42.

Phillip H. Miller. 1999. *Strong Generative Capacity: The Semantics of Linguistic Formalism*. Stanford, CA: CSLI Publications.

Mehryar Mohri. 1997. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311.

Max Nelson, Hossep Dolatian, Jonathan Rawski, and Brandon Prickett. 2020. Probing rnn encoder-decoder generalization of subregular functions using reduplication. In *Proceedings of the Society for Computation in Linguistics 2020*, pages 167–178.

Amanda Payne, Mai Ha Vu, and Jeffrey Heinz. 2016. A formal analysis of correspondence theory. In *Proceedings of the Annual Meetings on Phonology*.

Geoffrey K. Pullum. 2007. The evolution of model-theoretic frameworks in linguistics. In *Model-Theoretic Syntax at 10*, pages 1–10, Dublin, Ireland.

Stephan Rabanser, Oleksandr Shchur, and Stephan Günnemann. 2017. Introduction to tensor decompositions and their applications in machine learning. *arXiv preprint arXiv:1711.10781*.

Jonathan Rawski. 2019. Tensor product representations of subregular formal languages. In *Proceedings of the International Joint Conference on Artificial Intelligence workshop on Neural-Symbolic Learning and Reasoning*, pages 36–42.

Jonathan Rawski, Hossep Dolatian, Jeffrey Heinz, and Eric Raimy. 2023. Regular and polyregular theories of reduplication. *Glossa: a journal of general linguistics*, 8(1).

Brian Roark and Richard Sproat. 2007. *Computational Approaches to Morphology and Syntax*. Oxford University Press, Oxford.

James Rogers. 1998. *A descriptive approach to language-theoretic complexity*. CSLI Publications Stanford, CA.

Taisuke Sato. 2017. Embedding tarskian semantics in vector spaces. In *Workshops at the Thirty-First AAAI Conference on Artificial Intelligence*.

Luciano Serafini and Artur S d'Avila Garcez. 2016. Learning and reasoning with logic tensor networks. In *Conference of the Italian Association for Artificial Intelligence*, pages 334–348. Springer.

Paul Smolensky. 1990. Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial intelligence*, 46(1-2):159–216.

Paul Smolensky and Géraldine Legendre. 2006. *The Harmonic Mind: From Neural Computation to Optimality-Theoretic Grammar*, volume Volume I: Cognitive Architecture. MIT Press.

Lena Strobl, William Merrill, Gail Weiss, David Chiang, and Dana Angluin. 2023. Transformers as recognizers of formal languages: A survey on expressivity. *arXiv preprint arXiv:2311.00208*.

Whitney Tabor. 2009. A dynamical systems perspective on the relationship between symbolic and non-symbolic computation. *Cognitive neurodynamics*, 3(4):415–427.

Wolfgang Thomas. 1997. Languages, automata, and logic. In *Handbook of Formal Languages*, volume 3, chapter 7. Springer.

Sarah Winter. 2021. Decision problems for origin-close top-down tree transducers. In *46th International Symposium on Mathematical Foundations of Computer Science (MFCS 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2014. Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575*.