Linear Transduction Grammars and Zipper Finite-State Transducers

Markus Saers and **Dekai Wu**
Human Language Technology Center
Dept. of Computer Science and Engineering
Hong Kong University of Science and Technology
Hong Kong
{masaers|dekai}@cs.ust.hk

Abstract

We examine how the recently explored class of linear transductions relates to finite-state models. Linear transductions have been neglected historically, but gained recent interest in statistical machine translation modeling, due to empirical studies demonstrating that their attractive balance of generative capacity and complexity characteristics lead to improved accuracy and speed in learning alignment and translation models. Such work has until now characterized the class of linear transductions in terms of either (a) linear inversion transduction grammars (LITGs) which are linearized restrictions of inversion transduction grammars or (b) linear transduction grammars (LTGs) which are bilingualized generalizations of linear grammars. In this paper, we offer a new alternative characterization of linear transductions, as relating four finite-state languages to each other. We introduce the devices of zipper finite-state automata (ZFSAs) and zipper finite-state transducers (ZFSTs) in order to construct the bridge between linear transductions and finite-state models.

1 Introduction

Linear transductions are a long overlooked class of transductions positioned between finite-state transductions and inversion transductions in terms of complexity. In the Aho–Ullman hierarchy, linear transductions are those that can be generated by SDTGs[1] of rank 1, but that is about all that is said about them.

[1]Syntax-directed transduction grammars or SDTGs (Lewis and Stearns, 1968; Aho and Ullman, 1972) have also been referred to recently in the statistical machine translation subcommunity as synchronous context-free grammars.

Recently, however, linear transduction grammars (LTGs) have been shown to be both effective and efficient for learning word alignments in statistical machine translation models (Saers et al., 2010b; Saers et al., 2010a). LTGs can align words more accurately than FSTs since they allow words to be reordered, and yet alignment and training complexity is two orders of magnitude lower than with ITGs (Wu, 1997).

The added efficiency means that LTGs can be learned directly from parallel corpora rather than relying on external word alignment tools for *a priori* annotation. The added efficiency does, however, come at a price in expressivity, and it is vital to understand the nature of this trade-off. The automaton/transducer perspective of linear transductions described in this paper offers another vector towards understanding the properties of linear transductions.

Thus far, such work has not characterized the class of linear transductions in terms of finite-state models. Saers et al. (2010b) define linear transductions in terms of linear inversion transduction grammars (LITGs), which are inversion transduction grammars with a linear restriction. Alternatively, Saers et al. (2010a) introduce linear transduction grammars (LTGs), which are the natural bilingual generalization of linear grammars, and show that they define the same class of linear transductions as LITGs.

In this paper, we offer a new alternative characterization of linear transductions based on finite-state models. We will start by giving a definition of LTGs, as the principal mechanism for generating linear transductions (Section 2). As an intermediate step, we will note that linear languages (which are related by linear transductions) can be handled by FSTs under some conditions: we treat linear languages as two finite-state languages dependent on each other, by introducing the device of zipper finite-state automata (Section 3). We then general-

Proceedings of Recent Advances in Natural Language Processing, pages 640–647,
Hissar, Bulgaria, 12-14 September 2011.

$$
\begin{aligned}
&G = \langle \{S, F\}, \Sigma, \Delta, S, R \rangle \text{ such that} \\
&\Sigma = \{\text{b}, \text{l}, \text{sandwich}, \text{t}, \text{-}\}, \\
&\Delta = \{\text{bacon}, \text{bread}, \text{lettuce}, \text{mayonnaise}, \text{tomato}\}, \\
&R = \left\{ \begin{array}{l}
S \rightarrow \epsilon/\text{bread } F \text{ sandwich}/\text{bread}, \\
F \rightarrow \text{b}/\epsilon \; F \; \epsilon/\text{bacon}, \\
F \rightarrow \text{l}/\text{lettuce } F, \\
F \rightarrow \text{t}/\text{tomato } F, \\
F \rightarrow \text{-}/\epsilon, \\
F \rightarrow \text{-}/\text{mayonnaise}
\end{array} \right\}
\end{aligned}
$$

(a) Linear transduction grammar

$$
\begin{aligned}
S &\underset{G}{\Rightarrow} \epsilon/\text{bread } F \text{ sandwich}/\text{bread} \\
&\underset{G}{\Rightarrow} \text{b}/\text{bread } F \text{ sandwich}/\text{bacon bread} \\
&\underset{G}{\Rightarrow} \text{b l}/\text{bread lettuce } F \text{ sandwich}/\text{bacon bread} \\
&\underset{G}{\Rightarrow} \text{b l t}/\text{bread lettuce tomato } F \text{ sandwich}/\text{bacon bread} \\
&\underset{G}{\Rightarrow} \text{b l t - sandwich}/\text{bread lettuce tomato mayonnaise bacon bread}
\end{aligned}
$$

(b) Generation

Figure 1: A linear transduction grammar (a) generating a bistring (b). The transduction defined establishes the concept "BLT-sandwich" and the ordered components of its realization: bacon, lettuce and tomato (with optional mayonnaise) sandwiched between two slices of bread.

ize this to introduce zipper finite-state transducers, treating linear transductions as relating two linear languages to each other (Section 4). Since linear languages relate two finite-state languages to each other, and linear transductions relate two linear languages to each other, linear transductions can be said to relate four finite-state languages to each other.

2 Linear transduction grammars

A linear transduction grammar (LTG) is an inversion transduction grammar (ITG) or syntax-directed transduction grammar (SDTG), of rank 1, which means that any rule may produce at most one nonterminal, eliminating any branching. Figure 1 contains an example of an LTG, and how it generates a bistring.

Definition 1 A linear transduction grammar (LTG) over languages L_1 and L_2 is a tuple:

$$G = \langle N, \Sigma, \Delta, S, R \rangle$$

where N is a finite nonempty set of nonterminal symbols, Σ is a finite nonempty set of L_1 symbols, Δ is a finite nonempty set of L_2 symbols, $S \in N$ is the designated start symbol and R is a finite nonempty set of production rules on the forms:

$$
\begin{aligned}
A &\rightarrow {}^{a}\!/\!_{x} \; B \; {}^{b}\!/\!_{y} \\
A &\rightarrow {}^{a}\!/\!_{x}
\end{aligned}
$$

where the nonterminals $A, B \in N$ and the biterminals ${}^{a}\!/\!_{x}, {}^{b}\!/\!_{y} \in \Sigma^* \times \Delta^*$.

Definition 2 The rules in an LTG $G = \langle N, \Sigma, \Delta, S, R \rangle$ define a binary relation $\underset{G}{\Rightarrow}$ over $(\Sigma^* \times \Delta^*)(N \cup (\Sigma^* \times \Delta^*))(\Sigma^* \times \Delta^*)$ such that:

$$
\begin{aligned}
{}^{a}\!/\!_{w} \, A \, {}^{d}\!/\!_{z} &\underset{G}{\Rightarrow} {}^{ab}\!/\!_{wx} \, B \, {}^{cd}\!/\!_{yz} \quad \text{iff} \quad A \rightarrow {}^{b}\!/\!_{x} \, B \, {}^{c}\!/\!_{y} \in R \\
{}^{a}\!/\!_{w} \, A \, {}^{d}\!/\!_{z} &\underset{G}{\Rightarrow} {}^{abd}\!/\!_{wxz} \quad \text{iff} \quad A \rightarrow {}^{b}\!/\!_{x} \in R
\end{aligned}
$$

Note that both the biterminal expressions ${}^{a}\!/\!_{w} \; {}^{b}\!/\!_{x}$ and ${}^{ab}\!/\!_{wx}$ can designate the translation between the terminal strings ab and wx. The reflexive transitive closure of this relation can be used to define the transduction generated by an LTG as the set of bistrings that can be generated from the grammar's start symbol.

Definition 3 The transduction generated by the LTG $G = \langle N, \Sigma, \Delta, S, R \rangle$ is:

$$T(G) = \left\{ \langle a, x \rangle \middle| S \overset{*}{\underset{G}{\Rightarrow}} {}^{a}\!/\!_{x} \right\} \cap (\Sigma^* \times \Delta^*)$$

Even though no normal form is given in Aho and Ullman (1972) for LTGs or SDTGs of rank 1, it is useful to have such a normal form. In this work we will adopt the following normal form for LTGs.

Definition 4 An LTG in normal form is an LTG where the rules are constrained to have one of the forms:

$$A \to {}^{a}\!/\!_{x'} \;\; B \;\; {}^{b'}\!/\!_{y'} \qquad A \to {}^{a'}\!/\!_{x} \;\; B \;\; {}^{b'}\!/\!_{y'}$$
$$A \to {}^{a'}\!/\!_{x'} \;\; B \;\; {}^{b}\!/\!_{y'} \qquad A \to {}^{a'}\!/\!_{x'} \;\; B \;\; {}^{b'}\!/\!_{y}$$
$$A \to {}^{\epsilon}\!/\!_{\epsilon}$$

where $A, B \in N$, $a, b \in \Sigma$, $a', b' \in \Sigma \cup \{\epsilon\}$, $x, y \in \Delta$ and $x', y' \in \Delta \cup \{\epsilon\}$.

That is: only rules where at least one terminal symbol is produced together with a nonterminal symbol, and rules where the empty bistring is produced, are allowed. The "primed" symbols are allowed to be the empty string, whereas the others are not. It is possible to construct an LTG in normal form from an arbitrary LTG in the same way that a linear grammar (LG) is normalized.

Theorem 1. *Grammars of type* LTG *and type* LTG *in normal form generate the same class of transductions.*

Proof. Given an LTG $G = \langle N, \Sigma, \Delta, S, R \rangle$, we can construct an LTG in normal form $G' = \langle N', \Sigma, \Delta, S, R' \rangle$ that generates the same language. For every rule in R we can produce a series of corresponding rules in R'. We start by removing useless nonterminals, rules where one nonterminal rewrites into another nonterminal only. This can be done in the same way as for SDTGs, see Aho and Ullman (1972). The rules can then be recursively shortened until they are in normal form.

If the rule $A \to {}^{a}\!/\!_{x} \;\; B \;\; {}^{c}\!/\!_{z}$ is not in normal form, it can be rephrased as two rules:

$$A \to {}^{a}\!/\!_{x\,1} \;\; \bar{B} \;\; {}^{c}\!/\!_{z\,m}$$
$$\bar{B} \to {}^{a}\!/\!_{x\,2} \cdots {}^{a}\!/\!_{x\,n} \;\; B \;\; {}^{c}\!/\!_{z\,1} \cdots {}^{c}\!/\!_{z\,m-1}$$

where $\bar{B}$ is a newly created unique nonterminal, n is the length of the biterminal ${}^{a}\!/\!_{x}$ and m is the length of the biterminal ${}^{c}\!/\!_{z}$. The first rule is in normal form by definition. The second rule can be subjected to the same procedure until it is in normal form. Having either ${}^{a}\!/\!_{x}$ or ${}^{c}\!/\!_{z}$ be empty does not affect the results of the procedure, and since we started by eliminating useless rules, one of them is guaranteed to be nonempty.

If the rule $A \to {}^{b}\!/\!_{y}$ is not in normal form (meaning that ${}^{b}\!/\!_{y}$ is nonempty), it can be replaced by two rules:

$$A \to {}^{b}\!/\!_{y\,1} \;\; \bar{B}$$
$$\bar{B} \to {}^{b}\!/\!_{y\,2} \cdots {}^{b}\!/\!_{y\,n}$$

where $\bar{B}$ is a newly created unique nonterminal, and n is the length of the biterminal ${}^{b}\!/\!_{y}$. The set of nonterminals N' is the old set N in union with the set of all nonterminals that were created when R' was constructed.

Whenever there is a production in G such that:

$${}^{a}\!/\!_{w} \;\; A \;\; {}^{d}\!/\!_{z} \underset{G}{\Rightarrow} {}^{ab}\!/\!_{wx} \;\; B \;\; {}^{cd}\!/\!_{yz}$$

or

$${}^{a}\!/\!_{x} \;\; A \;\; {}^{c}\!/\!_{z} \underset{G}{\Rightarrow} {}^{abc}\!/\!_{xyz}$$

There is, by construction, a sequence of productions in G' such that:

$${}^{a}\!/\!_{w} \;\; A \;\; {}^{d}\!/\!_{z} \overset{*}{\underset{G'}{\Rightarrow}} {}^{ab}\!/\!_{wx} \;\; B \;\; {}^{cd}\!/\!_{yz}$$

or

$${}^{a}\!/\!_{x} \;\; A \;\; {}^{c}\!/\!_{z} \overset{*}{\underset{G'}{\Rightarrow}} {}^{abc}\!/\!_{xyz}$$

This means that G' is capable of generating any string that G can generate, giving us the inequality:

$$L(G) \subseteq L(G')$$

Since the normal form constitutes a restriction, we also know that:

$$L(G') \subseteq L(G)$$

Which leads us to conclude that:

$$L(G) = L(G') \qquad \square$$

For statistical machine translation applications, LTGs can be made weighted or stochastic (Saers et al., 2010b; Saers et al., 2010a) in the same way as ITGs Wu (1997).

3 Linear languages revisited

In this section we will take a look at the connection between linear languages (LLs) and FSTs, and leverage the relationship to define a new type of automaton to handle LLs. The new class of automata is referred to as zipper finite-state automata (ZFSAs), which will replace one-turn pushdown automata (Ginsburg and Spanier, 1966, 1-PDAs) and nondeterministic two-tape automata (Rosenberg, 1967, 2-NDAs) as the principal machine for handling linear languages. This is mainly to facilitate the move into the bilingual domain, and offers nothing substantially new.

Ginsburg and Spanier (1966, Theorem 6.1) show that a linear language can be seen as the input to an FST concatenated with the reverse of its output. Rosenberg (1967, Theorems 9 and 10) shows that any linear grammar can be said to generate the concatenation of the first tape from a 2-NDA with the reverse of the second. Instead of giving the original theorems, we will give two lemmas in the spirit of the previous works.

Lemma 2. *For every one-restricted finite-state transducer (*1-FST*) M there is an* LG *in normal form that generates the language* $\{ab^{\leftarrow}|\langle a,b\rangle \in T(M)\}$.[2]

Proof. Given that $M = \langle Q, \Sigma, \Delta, q_0, F, \delta\rangle$ is a 1-FST, we can construct an LG in normal form $G = \langle Q, \Sigma \cup \Delta, q_0, R\rangle$ where

$$R = \big\{q \to a\ q'\ b \big| \langle q, a, b, q'\rangle \in \delta\big\} \cup \{q \to \epsilon | q \in F\}$$

where $q, q' \in Q$, $a \in \Sigma \cup \{\epsilon\}$ and $b \in \Delta \cup \{\epsilon\}$. Whenever there is a transition sequence with M such that:

$$\begin{aligned}\langle q_0, a, b\rangle &= \langle q_0, a_1 \dots a_n, b_1 \dots b_n\rangle \\ &\vdash_M \langle q_1, a_2 \dots a_n, b_2 \dots b_n\rangle \\ &\vdash_M^* \langle q_{n-1}, a_n, b_n\rangle \\ &\vdash_M \langle q_n, \epsilon\rangle\end{aligned}$$

where $q_i \in Q$, $a_i \in \Sigma \cup \{\epsilon\}$, $a \in \Sigma^*$, $b_i \in \Delta \cup \{\epsilon\}$ and $b \in \Delta^*$ for all i, and where the state q_n is a member of F, there is, by construction, a derivation with G such that:

$$\begin{aligned}q_0 &\underset{G}{\Rightarrow} a_1 q_1 b_1 \\ &\underset{G}{\overset{*}{\Rightarrow}} a_1 \dots a_n q_n b_n \dots b_1 \\ &\underset{G}{\Rightarrow} a_1 \dots a_n b_n \dots b_1 = ab^{\leftarrow}\end{aligned}$$

Thus: whenever the bistring $\langle a, b\rangle$ is a member of $T(M)$, the string $ab^{\leftarrow}$ is a member of $L(G)$. By construction, G cannot generate any other strings. We thus conclude that

$$L(G) = \{ab^{\leftarrow}|\langle a,b\rangle \in T(M)\} \qquad \square$$

Lemma 3. *For every* LG *in normal form (G), there exists a* 1-FST *(M) such that, for all string $s \in L(G)$, there exists a partition $s = ab^{\leftarrow}$ such that $\langle a, b\rangle \in T(M)$.*

Proof. Given that $G = \langle N, \Sigma, S, R\rangle$ is an LG in normal form, we can construct a 1-FST $M = \langle N, \Sigma, \Sigma, S, F, \delta\rangle$ where:

$$\begin{aligned}F &= \{A | A \to \epsilon \in R\}, \\ \delta &= \{\langle A, a, b, B\rangle | A \to a\ B\ b \in R\}\end{aligned}$$

where $A, B \in N$ and $a, b \in \Sigma \cup \{\epsilon\}$. Whenever there is a derivation with G such that:

$$\begin{aligned}S &\underset{G}{\Rightarrow} a_1 X_1 b_1 \\ &\underset{G}{\overset{*}{\Rightarrow}} a_1 \dots a_n X_n b_n \dots b_1 \\ &\underset{G}{\Rightarrow} a_1 \dots a_n b_n \dots b_1 = ab^{\leftarrow}\end{aligned}$$

there is, by definition, a sequence of transitions with M such that:

$$\begin{aligned}\langle S, a, b\rangle &= \langle S, a_1 \dots a_n, b_1 \dots b_n\rangle \\ &\vdash_M \langle X_1, a_2 \dots a_n, b_2 \dots b_n\rangle \\ &\vdash_M^* \langle X_{n-1}, a_n, b_n\rangle \\ &\vdash_M \langle X_n, \epsilon, \epsilon\rangle\end{aligned}$$

(where $q_i \in Q$, $a_i \in \Sigma \cup \{\epsilon\}$, $a \in \Sigma^*$, $b_i \in \Delta \cup \{\epsilon\}$ and $b \in \Delta^*$ for all i) and the state X_n is by definition a member of F. Thus: whenever G generates $ab^{\leftarrow}$, M can recognize $\langle a, b\rangle$. By construction, M cannot recognize any other bistrings. We thus conclude that

$$T(M) = \{\langle a,b\rangle | ab^{\leftarrow} \in L(R)\} \qquad \square$$

[2] Where $b^{\leftarrow}$ is used to mean the reverse of b.

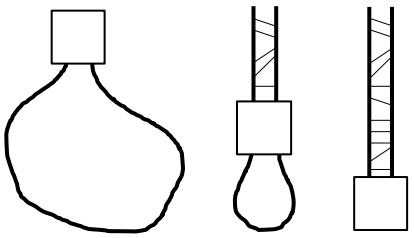

Figure 2: A zipper finite-state automata relates the two parts of a string to each other, and define the partitioning of the string at the same time.

There is a discrepancy between FSTs and linear languages in that every string in the language has to be partitioned into two strings before the FST can process them. Naturally, the number of ways to partition a string is proportional to its length. Naïvely trying all possible partitions would take $\mathcal{O}(n^3)$ time ($\mathcal{O}(n)$ partitions and $\mathcal{O}(n^2)$ time to run the FST on each string pair), which is equal to CFGs. If linear languages are as time-consuming to process as CFLs, we might as well use the more expressive language class. If, however, the partition point could be found as a part of the analysis process rather than conjectured *a priori*, the process would be faster than CFGs.

It is possible to reinterpret the transition relation defined by an FST such that it reads from both tapes, rather than reads from one and writes to the other. We define this relation as:

$$\langle q, a\alpha, \beta b\rangle \vdash_{M,r} \langle q', \alpha, \beta\rangle \quad \text{iff} \quad \langle q, a, b, q'\rangle \in \delta$$

where $q, q' \in Q$, $a \in \Sigma$, $b \in \Delta$, $\alpha \in \Sigma^*$ and $\beta \in \Delta^*$. Using this interpretation of the FST M (designated M' under this reinterpretation) we have that:

$$\langle \alpha, \beta^{\leftarrow}\rangle \in T(M) \quad \text{iff} \quad \langle \alpha, \beta\rangle \in T(M')$$

which, by lemmas 2 and 3, means that the concatenation of α and β over the entire transduction constitutes a linear language. This is the intuition behind zipper finite-state automata. By constructing a string $\gamma \in (\Sigma \cup \Delta)^*$ such that $\gamma = \alpha\beta$, we can rewrite the reinterpreted FST relation as:

$$\langle q, a\gamma b\rangle \vdash_{M',r} \langle q', \gamma\rangle \quad \text{iff} \quad \langle q, a, b, q'\rangle \in \delta$$

which define a linear language over $(\Sigma \cup \Delta)^*$. The partitioning of the string is also implicitly defined since the automaton will end up somewhere in the original string, defining the place of partitioning that makes the two parts related (or concluding that they are not, and that the string is not a member of the language defined by the automaton). The attribute "zipper" comes from the visualization, where the control of the automaton slides down two ends of the tape until it reaches the bottom after having drawn all connections between the two parts of the tape—like a zipper (see Figure 2). Again, this is merely a reinterpretation of previous work. The idea of a dedicated automaton to process a single tape containing strings from a linear language with finite control (as opposed to using a stack as the 1-PDAs do, or partitioning the tapes as 2-NDAs strictly speaking have to do) is not new. Nagy (2008) presents $5' \rightarrow 3'$ sensing Watson-Crick finite automata which are used to process DNA strings, and Loukanova (2007) presents nondeterministic finite automata to handle linear languages. Our reinterpretation is made to facilitate the transition into the bilingual domain.

Definition 5 A zipper finite-state automaton (ZFSA) is a tuple:

$$M = \langle Q, \Sigma, q_0, F, \delta\rangle$$

where Q is a finite nonempty set of states, Σ is a finite set of symbols, $q_0 \in Q$ is the start state, $F \subseteq Q$ is a set of accepting states and $\delta \subseteq Q \times \Sigma^* \times \Sigma^* \times Q$ is a finite set of transitions. Transitions define a binary relation over $Q \times \Sigma^*$ such that:

$$\langle q, \alpha\gamma\beta\rangle \vdash_M \langle q', \gamma\rangle \quad \text{iff} \quad \langle q, \alpha, \beta, q'\rangle \in \delta$$

where $q, q' \in Q$ and $\alpha, \beta, \gamma \in \Sigma^*$.

Lemma 4. *Every* FST *can be expressed as a* ZFSA.

Proof. Let $M = \langle Q, \Sigma, \Delta, q_0, F, \delta\rangle$ be an FST, and let $M' = \langle Q, \Sigma \cup \Delta, q_0, F, \delta\rangle$ be the corresponding ZFSA. The only differences are that M' uses the union of the two alphabets that M transduces between, and that the interpretation of the relation defined by δ is different in M and M'. □

Lemma 5. *Every* ZFSA *can be expressed as an* FST.

Proof. Let $M = \langle Q, \Sigma, q_0, F, \delta\rangle$ be a ZFSA, and let $M' = \langle Q, \Sigma, \Sigma, q_0, F, \delta\rangle$ be the corresponding FST transducing within the same alphabet. The only differences are that M' uses two copies of

$$
\begin{aligned}
M &= \langle Q, \Sigma, \Delta, q_S, \{q'\}, \delta\rangle \text{ such that}\\
Q &= \{q_S, q_F, q'\},\\
\Sigma &= \{\text{b}, \text{l}, \text{sandwich}, \text{t}, \text{-}\},\\
\Delta &= \{\text{bacon}, \text{bread}, \text{lettuce}, \text{mayonnaise}, \text{tomato}\},\\
\delta &= \left\{\begin{array}{l}
\langle q_S, \epsilon, \text{bread}, \text{sandwich}, \text{bread}, q_F\rangle,\\
\langle q_F, \text{b}, \epsilon, \epsilon, \text{bacon}, q_F\rangle,\\
\langle q_F, \text{l}, \text{lettuce}, \epsilon, \epsilon, q_F\rangle,\\
\langle q_F, \text{t}, \text{tomato}, \epsilon, \epsilon, q_F\rangle,\\
\langle q_F, \text{-}, \epsilon, \epsilon, \epsilon, q'\rangle\\
\langle q_F, \text{-}, \text{mayonnaise}, \epsilon, \epsilon, q'\rangle
\end{array}\right\}
\end{aligned}
$$

(a) Zipper finite-state transducer

$$
\begin{aligned}
&\langle q_S, \text{b l t -}\ \text{sandwich}, \text{bread lettuce tomato mayonnaise bacon bread}\rangle\\
&\vdash_M \langle q_F, \text{b l t -}, \text{lettuce tomato mayonnaise bacon}\rangle\\
&\vdash_M \langle q_F, \text{l t -}, \text{lettuce tomato mayonnaise}\rangle\\
&\vdash_M \langle q_F, \text{t -}, \text{tomato mayonnaise}\rangle\\
&\vdash_M \langle q_F, \text{-}, \text{mayonnaise}\rangle\\
&\vdash_M \langle q', \epsilon, \epsilon\rangle
\end{aligned}
$$

(b) Recognition

Figure 3: A zipper finite-state transducer (a) recognizing a bistring (b). This is the same bistring that was generated in Figure 1.

the same alphabet (Σ), and that the interpretation of the relation defined by δ is different in M and M'. □

Theorem 6. FSTs *in recognition mode are equivalent to* ZFSAs*.*

Proof. Follows from Lemmas 4 and 5. □

Theorem 7. *The class of languages recognized by* ZFSAs *is the class of linear languages.*

Proof. From Lemmas 2 and 3 we have that FSTs generate linear languages, and from theorem 6 we have that ZFSAs are equivalent to FSTs. □

To recognize with a ZFSA is as complicated as recognizing with an FST, which can be done in $\mathcal{O}(n^2)$ time. Since we are effectively equating a transduction with a language, it is helpful to instead consider this as "finite-state in two dimensions." For the finite-state transduction, this is easy, since it relates two finite-state languages to each other. For the linear languages it takes a little more to consider them as languages that internally relate one part of every string to the other part of that string. The key point is that they are both relating something that is in some sense finite-state to something else that is also finite-state.

4 Zipper finite-state transducers

Having condensed a finite-state relation down to a language, we can relate two such languages to each other. This is what zipper finite-state transducers (ZFSTs) do. If linear languages relate one part of every string to the other, linear transductions relate these two parts to the two parts of all the strings in another linear language. There are in all four kinds of entities involved, $\langle a, b\rangle \in L_1$ and $\langle x, y\rangle \in L_2$, and linear transduction have to relate them all to each other. We claim that this is what LTGs do, and in this section we will see that the transducer class for linear languages, ZFSTs, is equivalent to LTGs. An example of a ZFST can be found in Figure 3.

Definition 6 A ZFST over languages L_1 and L_2 is a tuple:

$$M = \langle Q, \Sigma, \Delta, q_0, F, \delta\rangle$$

where Q is a finite nonempty set of states, Σ is a finite nonempty set of L_1 symbols, Δ is a

finite nonempty set of L_2 symbols, $q_0 \in Q$ is the designated start state, $F \subseteq Q$ is a set of accepting states and:

$$\delta \subseteq Q \times \Sigma^* \times \Delta^* \times \Sigma^* \times \Delta^* \times Q$$

is a finite set of transitions. The transitions define a binary relation over $Q \times \Sigma^* \times \Delta^*$ such that:

$$\langle q, abc, xyz\rangle \vdash_M \langle q', b, y\rangle \quad \text{iff} \quad \langle q, a, x, c, z, q'\rangle \in \delta$$

where $q, q' \in Q$, $a, b, c \in \Sigma^*$ and $x, y, z \in \Delta^*$.

We know that ZFSTs relate linear languages to each other, they are defined to do so, and we conjecture that LTGs relate linear languages to each other. By proving that ZFSTs and LTGs handle the same class of transductions we can assert that LTGs do indeed generate a transduction relation between linear languages.

Lemma 8. *For every* LTG *there is a* ZFST *that recognizes the language generated by the* LTG.

Proof. Let $G = \langle N, \Sigma, \Delta, S, R\rangle$ be an LTG, and let $M = \langle N', \Sigma, \Delta, S, \{S'\}, \delta\rangle$ be the corresponding ZFST where S' is a unique final state, $N' = N \cup \{S'\}$ and:

$$\delta = \big\{\langle A, a, x, c, z, B\rangle \big| A \to {}^{a}\!/\!_{x}\ B\ {}^{c}\!/\!_{z} \in R\big\} \cup \big\{\langle A, b, y, \epsilon, \epsilon, S'\rangle \big| A \to {}^{b}\!/\!_{y} \in R\big\}$$

where $A, B \in N$, $a, b, c \in \Sigma^*$ and $x, y, z \in \Delta^*$. Whenever there is a derivation with G such that:

$$\begin{aligned} S &\underset{G}{\Rightarrow} {}^{a_1}\!/\!_{x_1}\ A_1\ {}^{c_1}\!/\!_{z_1} \\ &\overset{*}{\underset{G}{\Rightarrow}} {}^{a_1}\!/\!_{x_1} \ldots {}^{a_n}\!/\!_{x_n}\ A_n\ {}^{c_n}\!/\!_{z_n} \ldots {}^{c_1}\!/\!_{z_1} \\ &\underset{G}{\Rightarrow} {}^{a_1}\!/\!_{x_1} \ldots {}^{a_n}\!/\!_{x_n}\ {}^{b}\!/\!_{y}\ {}^{c_n}\!/\!_{z_n} \ldots {}^{c_1}\!/\!_{z_1} \end{aligned}$$

(where $S, A_i \in N$, $a_i, b_i \in \Sigma^*$ and $x_i, y_i \in \Delta^*$ for all i),[3] we have, by construction, a sequence of transitions in M that takes it from an initial configuration with the generated bistring to an accepting configuration:

$$\begin{aligned} &\langle S, a_1 \ldots a_n b c_n \ldots c_1, x_1 \ldots x_n y z_n \ldots z_1\rangle \\ \vdash_M &\langle A_1, a_2 \ldots a_n b c_n \ldots c_2, x_2 \ldots x_n y z_n \ldots z_2\rangle \\ \vdash^*_M &\langle A_n, b, y\rangle \\ \vdash_M &\langle S', \epsilon, \epsilon\rangle \end{aligned}$$

This means that M can recognize all strings generated by G. By construction, M cannot recognize any other strings. We thus conclude that

$$T(M) = T(G) \qquad \square$$

Lemma 9. *For every* ZFST*, there is an* LTG *that generates the transduction recognized by the* ZFST.

Proof. Let $M = \langle Q, \Sigma, \Delta, q_0, F, \delta\rangle$ be a ZFST, and let $G = \langle Q, \Sigma, \Delta, q_0, R\rangle$ be the corresponding LTG where:

$$R = \big\{q \to {}^{a}\!/\!_{x}\ \ q'\ \ {}^{b}\!/\!_{y} \ \big|\langle q, a, x, b, y, q'\rangle \in \delta\big\} \cup \big\{q \to {}^{\epsilon}\!/\!_{\epsilon} \ \big|q \in F\big\}$$

where $q, q' \in Q$, $a, b, c \in \Sigma^*$ and $x, y, z \in \Delta^*$. For every bistring that M can recognize:

$$\begin{aligned} &\langle q_0, a_1 \ldots a_n b_n \ldots b_1, x_1 \ldots x_n y_n \ldots y_1\rangle \\ \vdash_M &\langle q_1, a_2 \ldots a_n b_n \ldots b_2, x_2 \ldots x_n y_n \ldots y_2\rangle \\ \vdash^*_M &\langle q_n, \epsilon, \epsilon\rangle \end{aligned}$$

(where $q_i \in Q$, $a_i, b_i \in \Sigma^*$ and $x_i, y_i \in \Delta^*$ for all i,[4] and where $q_n \in F$), we have, by construction, a derivation with G that generates that bistring:

$$\begin{aligned} q_0 &\underset{G}{\Rightarrow} {}^{a_1}\!/\!_{x_1}\ q_1\ {}^{b_1}\!/\!_{y_1} \\ &\overset{*}{\underset{G}{\Rightarrow}} {}^{a_1}\!/\!_{x_1} \ldots {}^{a_n}\!/\!_{x_n}\ q_n\ {}^{b_n}\!/\!_{y_n} \ldots {}^{b_1}\!/\!_{y_1} \\ &\underset{G}{\Rightarrow} {}^{a_1}\!/\!_{x_1} \ldots {}^{a_n}\!/\!_{x_n}\ {}^{b_n}\!/\!_{y_n} \ldots {}^{b_1}\!/\!_{y_1} \end{aligned}$$

This means that G can generate all strings that M can recognize. By construction, G cannot generate any other strings. We thus conclude that

$$T(G) = T(M) \qquad \square$$

Theorem 10. *The class of transductions generated by* LTG*s is the same as that recognized by* ZFST*s.*

Proof. Follows from lemmas 8 and 9. $\square$

[3] These i indices are not indicating individual symbols in a string, but different strings.

[4] Again, these indices do not refer to individual symbols in a string, but different strings.

5 Conclusion

We have examined how the class of linear transductions relates to finite-state models. Our analysis complements earlier characterizations of linear transductions in terms of LITGs (linearized restrictions of inversion transduction grammars) and LTGs (bilingualized generalizations of linear grammars). Our new alternative characterization has shown how linear transductions relate four finite-state languages to each other, with the aid of the devices zipper finite-state automata and transducers.

Acknowledgments

This work was funded by the Defense Advanced Research Projects Agency under GALE Contract Nos. HR0011-06-C-0023 and HR0011-06-C-0023, and the Hong Kong Research Grants Council (RGC) under research grants GRF621008, GRF612806, DAG03/04.EG09, RGC6256/00E, and RGC6083/99E. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the Defense Advanced Research Projects Agency. We would also like to thank the four anonymous reviewers, whose feedback made this a better paper.

References

Alfred V. Aho and Jeffrey D. Ullman. 1972. *The Theory of Parsing, Translation, and Compiling*. Prentice-Halll, Englewood Cliffs, NJ.

Seymour Ginsburg and Edwin H. Spanier. 1966. Finite-turn pushdown automata. *Society for Industrial and Applied Mathematics Journal on Control*, 4(3):429–453.

Philip M. Lewis and Richard E. Stearns. 1968. Syntax-directed transduction. *Journal of the Association for Computing Machinery*, 15(3):465–488.

Roussanka Loukanova. 2007. Linear context free languages. In Cliff Jones, Zhiming Liu, and Jim Woodcock, editors, *Theoretical Aspects of Computing – ICTAC 2007*, volume 4711 of *Lecture Notes in Computer Science*, pages 351–365. Springer Berlin/Heidelberg.

Benedek Nagy. 2008. On $5' \rightarrow 3'$ sensing Watson–Crick finite automata. In Max Garzon and Hao Yan, editors, *DNA Computing*, volume 4848 of *Lecture Notes in Computer Science*, pages 256–262. Springer Berlin/Heidelberg.

Arnold L. Rosenberg. 1967. A machine realization of the linear context-free languages. *Information and Control*, 10:175–188.

Markus Saers, Joakim Nivre, and Dekai Wu. 2010a. A systematic comparison between inversion transduction grammar and linear transduction grammar for word alignment. In *Proceedings of the 4th Workshop on Syntax and Structure in Statistical Translation*, pages 10–18, Beijing, China, August. Coling 2010 Organizing Committee.

Markus Saers, Joakim Nivre, and Dekai Wu. 2010b. Word alignment with stochastic bracketing linear inversion transduction grammar. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 341–344, Los Angeles, California, June. Association for Computational Linguistics.

Dekai Wu. 1997. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational Linguistics*, 23(3):377–403.