Chapter 22

Logical Perspectives on Strictly Local Transformations

Jane Chandlee and Steven Lindell

22.1 Introduction

The chapters in Part 2 of this volume have broadly demonstrated how phonological maps can be represented as graph transductions using first order (FO) logic. The fact that FO logic is sufficient for modeling these phonological maps points to an upper bound on the computational complexity of phonological grammars. In this chapter, however, we provide evidence that this bound is too high, meaning it may be the case that a logic more restrictive than FO will still be sufficient for a certain class of phonological maps. Specifically, we characterize local phonological maps as quantifier-free logical interpretations over strings with adjacency.

This result stems from previous work on phonological maps grounded in finite state automata. It has long been known that phonological maps describable with rewrite rules of the form A $\rightarrow$ B / C $\_\_$ D describe *regular relations*, provided the rule does not re-apply to the locus of its structural change (Johnson, 1972; Kaplan and Kay, 1994). More recent work has shown that at least with respect to local phonological maps, the class of regular relations is not restrictive enough. Chandlee (2014) defines *input strictly local (ISL) functions*, a class of string-to-string functions that is a proper subset of the regular relations, and demonstrates that they are

sufficient to model (noniterative) phonological maps with local triggers. This result is significant for at least two reasons. One, it provides a better characterization of the typology of local phonological maps. Two, the restrictive nature of ISL functions provides learnability advantages. Unlike the regular relations, the ISL functions are learnable from positive data (Chandlee *et al.*, 2014; Jardine *et al.*, 2014) in the sense of Gold (1967).

In this chapter we provide a logical characterization of the ISL functions and prove its equivalence to the automata-theoretic characterization given by Chandlee (2014) and Chandlee *et al.* (2014). For an example of the finite state characterization, consider the rule of postnasal obstruent voicing in 14, which is attested in Puyo Pungo Quechua (Pater, 2004).

(14) $[-\text{son}] \rightarrow [+\text{voice}] \;/\; [+\text{nasal}]\ \_\_$

(15) Puyo Pungo Quechua (Quechua; Pater, 2004)
/kampa/ $\mapsto$ [kamba] 'yours'

Figure 22.1 presents the finite state transducer (FST) for the postnasal obstruent voicing function. For readability, this FST is defined with a reduced alphabet of {V, D, T, N}, where V=vowel, D=voiced obstruent, T=voiceless obstruent, and N=nasal consonant. It has also been minimized, such that states 'V', 'T', and 'D' are collapsed with the start state. Lastly, single-symbol transition labels indicate identity mappings (e.g., X = X:X).

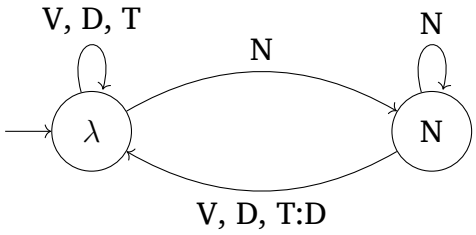

Figure 22.1: 2-ISL FST for postnasal obstruent voicing map

Given the input string 'kampa', the FST in Figure 22.1 would follow the path shown in 16. The input is read one segment at a time and the FST proceeds through its states according to the labeled transitions.

	Input:		k		a		m		p		a	
(16)	States:	λ	$\Rightarrow$	λ	$\Rightarrow$	λ	$\Rightarrow$	N	$\Rightarrow$	λ	$\Rightarrow$	λ
	Output:		k		a		m		b		a	

For the most part, the input is reproduced faithfully in the output. The crucial transition is from state N, which is reached anytime the FST encounters a nasal consonant. If the next segment is a voiceless obstruent, it is output as its voiced counterpart via the T:D transition.[1]

The fact that this map is describable with a FST proves that it is a regular relation. What establishes it further as an ISL function is that the FST in Figure 22.1 has certain properties not shared by FSTs generally. These properties enforce a type of short-term memory on the FST, such that at any given time it can only factor in the most recent input when determining what to output. What counts as 'most recent' is a parameter of a given ISL function, called its k-value. In the case of postnasal obstruent voicing, $k = 2$, since the needed information (i.e., a nasal followed by a voiceless obstruent) forms a contiguous substring of length 2.

Specifically, the properties that make the FST in Figure 22.1 an ISL FST are: 1) the states correspond to all possible input sequences of length up to $k - 1$, and 2) the transitions force the FST to always be in the state that represents the most recently read input sequence of length $k - 1$.[2] FSTs in general are not held to these requirements on states and transitions. Machines that have these restrictions are also called local machines of degree k (Sakarovitch, 2009).

The restrictive nature of the ISL class of functions that is reflected in its FST characterization also carries over to the logical characterization we present in this chapter. As in all of the case studies in the previous chapters, the postnasal obstruent voicing map can be modeled as a graph transduction using FO logic. However, we will show that it can in fact be modeled without using the full power of FO. More generally, our main result is that ISL functions (without null cycles) are **quantifier-free logical interpretations over strings with adjacency**.

In §22.2 we first provide the necessary preliminaries, and then in §22.3 we provide the intuition behind the proof by giving several examples of ISL phonological maps represented as quantifier-free logical interpretations.

[1] We put aside the fact that in Puyo Pugo Quechua this function only applies across a morpheme boundary (i.e., tautomorphemic sequences of a nasal and voiceless stop are permitted). Incorporating that added fact into the analysis would not change its computational classification: it would still be an ISL map, but it would be 3-ISL to include the morpheme boundary in the triggering environment.

[2] Again because the FST in the figure has been minimized, not all states are shown separately.

The proof of our main result is given in §22.4.

22.2 Preliminaries

22.2.1 Finite-state transducers

Let $|w|$ denote the length of a finite string. A deterministic finite-state transducer (FST) is a 5-tuple $(Q, q_0, \Sigma, \Gamma, \delta)$, where Q is a finite set of states, Σ and Γ are finite alphabets, $q_0 \in Q$ is the start state, and the function δ: $Q \times \Sigma \rightarrow \Gamma^* \times Q$ are the transitions. We break up $\delta(q, a)$ into its two components by letting δ_1: $Q \times \Sigma \rightarrow \Gamma^*$ be the output transitions and δ_2: $Q \times \Sigma \rightarrow Q$ the state transitions. A *subsequential* FST is a 6-tuple $(Q, q_0, \Sigma, \Gamma, \delta, \rho)$, where Q, q_0, Σ, Γ, and δ are defined as above and ρ: $Q \rightarrow \Gamma^*$ is a *final output function* that maps states to strings.

The subsequential FSTs used to model ISL functions have the following additional restrictions (Chandlee, 2014; Chandlee *et al.*, 2014):[3]

1. $Q = \Sigma^{<k}$ and $q_0 = \lambda$
2. $\delta_2(q, a) = S^{k-1}(qa)$, where $S^n(x)$ is the suffix of length n of a string x (with $S^n(x) = x$ when $|x| < n$).

22.2.2 Language theory

Let $P(w) = \{p \in \Sigma^* : w = ps$ for some suffix $s \in \Sigma^*\}$ be the set of all prefixes of w. If $w = ps$, then we can write $s = p^{-1}w$. For a set of strings S, let $P(S) = \bigcap\{P(w) : w \in S\}$ be the set of common prefixes of all strings in S. If S is a nonempty set of strings, then the longest common prefix $\bigwedge S$ is defined to be the longest prefix in $P(S)$.

For a function f and set of strings S, $f[S] = \{f(y) : y \in S\}$. We now define the notion of a tail of a string x with respect to a function f.

Definition 7. (Sakarovitch, 2009, p.692) For a function f: $\Sigma^* \rightarrow \Gamma^*$ and $x \in \Sigma^*$ let $f_x(y) = (\bigwedge f[x\Sigma^*])^{-1} f(xy)$ be the *tail* of f at x.

[3] See also Sakarovitch (2009, pg. 661-664) on *local functions*.

Informally, the tail of x is a function that maps all possible extensions y of x to y's unique contribution to the output string of xy. The uniqueness of this contribution is obtained by removing the longest common prefix of all output strings whose input string begins with x. Note that $\bigwedge f[x\Sigma^*]$ is always a prefix of $f(xy)$ for any y, so the use of inverse makes sense here.

It has been shown that f can be computed by a deterministic finite-state transducer iff its set of tails $\{f_x : x \in \Sigma^*\}$ is finite (Oncina and Garcia, 1991). And if its output depends only on the most recent k inputs, the function is k-input strictly local (k-ISL) (Chandlee, 2014).

Definition 8. A function f is ***k-local (k-ISL)*** if for all strings x of length $k-1$, $f_{ux} = f_x$ for all $u \in \Sigma^*$. Note that a k-local function is regular because it has at most $|\Sigma|^{k-1}$ tails (strings with same length $k-1$ suffix have the same tails).

A function from Σ^* to Γ^* is k-ISL iff it can be computed by a local automaton whose states correspond to $\Sigma^{<k}$, input strings of length less than k, where each transition between states is labeled with an input symbol from Σ and an output string from Γ^* (Chandlee, 2014; Chandlee *et al.*, 2014).

In order to ensure closure under composition among ISL functions, we need to bound the ratio between their input and output string lengths. The easiest way to enforce this is to forbid the automaton from having a null cycle—a closed circuit of transitions all of which output the empty string. However, there is a more elegant language theoretic way of expressing this.

Definition 9. A function $f\colon \Sigma^* \to \Gamma^*$ is ***finite-to-one*** if $f^{-1}(y)$ is always finite. I.e., for every $y \in \Gamma^*$, $|\{x : f(x) = y\}|$ is finite. Note that this is equivalent to saying that $f[L]$ is infinite whenever L is infinite.

An example of an ISL phonological map that is *not* a finite-to-one function would be one that reduces word-initial consonant clusters of any length down to a single consonant. In a rule-based formalism, this would be represented with a rule like $C \to \varnothing / CC_0 \_\_$, where C_0 abbreviates any string of zero or more consonants.[4] This rule maps the infinite set of strings

[4]In a constraint-based formalism like Optimality Theory, this same map can be represented with the constraint ranking *#CC $\gg$ MAX.

C^+V to the single string CV. Hence it is infinite-to-one.

Claim: Suppose a function f is computed by a minimal finite-state machine M. Then f is finite-to-one if and only if M has no null cycles.

Proof. Suppose M has a null cycle C labeled by v. Since M is minimal, every state is reachable, and so we can get to C from the start state via some string u, and then $f[uv^*] = \{f(u)\}$, which makes f infinite-to-one. On the other hand, if M has n states, every path of length n contains a cycle, so every sequence of n symbols must produce some output (since M does not contain a null cycle). So a string w must produce an output of length at least $|w| \div n$. If L is infinite, it contains arbitrarily long strings, thereby producing arbitrarily long output, and so $f[L]$ is infinite. □

22.2.3 Model theory

We use a special kind of structure to model strings with adjacency.

Definition 10. A *monadic* structure is a structure of the form $\langle D; f, ..., R, ...\rangle$ where D is a set (the domain), each $f\colon D \to D$ is a unary function, and each $R \subseteq D$ is a unary predicate.

Example: We use a linear monadic structure to represent strings. For example, $baab$ will be represented as $\langle\{1, 2, 3, 4\}; s, p, U_a, U_b\rangle$, where s is a successor function, p is a predecessor function, and $\{U_a, U_b\}$ partition the domain {1, 2, 3, 4} into the locations of the symbols a and b. This structure is shown in Figure 22.2.

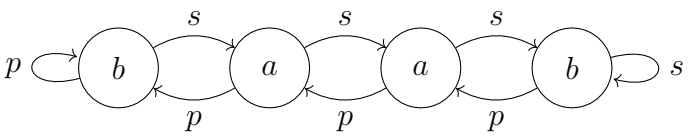

Figure 22.2: The structure for the string $baab$, where $U_{\mathsf{a}} = \{2, 3\}$ and $U_{\mathsf{b}} = \{1, 4\}$.

Note that s and p are inverses of each other, except when they reach their maximum and minimum, respectively, as shown in the figure. This is easily generalized to arbitrary finite strings over an arbitrary finite alphabet.

Definition 11. A Σ-structure is a tuple $\langle \{1, ..., n\}; s, p, \{U_\sigma : \sigma \in \Sigma\}\rangle$ where $s(i) = i+1$ (except $s(n) = n$) and $p(i) = i-1$ (except $p(1) = 1$), such that $\{U_\sigma : \sigma \in \Sigma\}$ is a partition of the domain $\{1, ..., n\}$. Each Σ-structure corresponds to a unique string in Σ^+ of length $n > 0$ whose i^{th} symbol is σ when $U_\sigma(i)$ is true.

We assume the reader is familiar with the basics of first-order interpretations on structures (see Courcelle, 1994; Engelfriet and Hoogeboom, 2001). Just as finite-state automata provide a mechanism for string to string transductions, logical formulas provide a way to define mappings from one structure to another, known as translations or interpretations. Although the general notion of an interpretation permits defining a mapping from any logical signature to another (i.e., a translation), we confine ourselves to mappings from strings to strings. In particular, we restrict ourselves to linear interpretations that produce an output consisting of m copies of the input for some fixed m, defined by vectors of formulas. We use superscript notation to denote various copies of formulas, variables, elements, or sets. For example, d^c represents the c^{th} copy of an element $d \in D$ for some domain D, and is really just an abbreviation for $\langle d, c\rangle \in D \times C$ for some copy set $C = \{1, ..., m\}$.

22.2.4 Polymorphic formulas: typed variables

Typed and extended terms

We elaborate terms from FO logic in two ways. First, the value of any term can be designated with a *type* from the copy set {1, ..., m}, so that every term has an assigned value and an assigned type. For example, x^c means the value x with type c. Polymorphic formulas can act on these typed terms so that different formulas can apply to different types.

Definition 12. A *polymorphic* formula $\overline{\chi}(x)$ is a vector $\langle \chi_1(x), ..., \chi_m(x)\rangle$ of ordinary (untyped) formulas, one for each possible type $i = 1, ..., m$ with the understanding that $\overline{\chi}(x^i) = \chi_i(x)$.

In this context, an ordinary formula $\chi(x)$ is just $\langle \chi(x), ...\chi(x)\rangle$, m copies of the same formula.

An *extended* term $t(x)$ is an ordinary typed term (a value and a type) or a conditional combination of two extended terms: if $\overline{\chi}(x)$ then $u(x)$ else $v(x)$; where $\overline{\chi}(x)$ is a polymorphic formula and $u(x)$ and $v(x)$ are extended

terms. Nested applications of this conditional construction allow us to form the more familiar *definitions by cases*, as in:

$$f(x) = \begin{cases} t_1(x) & \text{if } \chi_1(x) \\ \vdots & \vdots \\ t_{n-1}(x) & \text{if } \chi_{n-1}(x) \\ t_n(x) & \text{otherwise} \end{cases}$$

where each $t_j(x)$ is an extended term and each $\chi_j(x)$ a polymorphic formula (which may themselves contain extended terms). It is interpreted by giving priority to the order of the cases, which is to say the value of $f(x)$ is the value of $t_j(x)$ if $\chi_j(x)$ is satisfied and none of the previous conditions $\chi_1(x)$, ..., $\chi_{j-1}(x)$ are satisfied. This priority assumption guarantees extended terms always have unique values, just like ordinary terms. The otherwise clause is interpreted as always being true and ensures extended terms denote total functions, again like ordinary terms.

These conditionals are important for defining quantifier-free local functions, instead of defining a function by its graph. For example, the quantifier-free formula $\tau(x, y) \equiv s(y) = y$ defines the graph of a constant function $f(x) = y$ always equal to the last element, which is clearly non-local. But conditional term definitions of functions using only one variable always preserve locality.

Proposition 1. Any quantifier-free formula with extended terms is equivalent to an ordinary quantifier-free formula.

Proof. If $t(x)$ were to equal $u(x)$ when $\overline{\chi}(x)$ is true and $v(x)$ otherwise, just replace any occurrence ...$t(x)$... by $\overline{\chi}(x) \wedge ...u(x)... \vee \neg\overline{\chi}(x) \wedge ...v(x)...$. Since this conversion uses only Boolean combinations, every quantifier-free extended formula is equivalent to an ordinary quantifier-free formula. □

Using extended terms allows for simpler descriptions of polymorphic functions.

Example: Consider the interpretation that doubles every symbol (e.g., $baab \rightarrow bbaaaabb$), with $m = 2$. Define two adjacent copies x^1 and x^2 for every input element, defining the (polymorphic) successor (s) and predecessor (p) functions as: $s(x^1) = x^2$, $s(x^2) = (s(x))^1$ if $s(x) \neq x$ and x^2 otherwise; and $p(x^2) = x^1$, $p(x^1) = (p(x))^2$ if $p(x) \neq x$ and x^1 otherwise. Notice that the two copies of any element are placed next to each other.

Use $\alpha_a(x^1) \equiv \alpha_a(x^2) \equiv U_a(x)$ and $\alpha_b(x^1) \equiv \alpha_b(x^2) \equiv U_b(x)$ to say the copies are replicas of the originals.

picture must be changed

Figure 22.3: Illustration of an instance of the 2-copy interpretation, $baab \rightarrow bbaaaabb$

The generalized version is defined as follows.

Definition 13. An *m-copy interpretation* from Σ-structures to Γ-structures is a tuple

$$\langle \overline{\varphi}(x); \overline{\pi}(x), \overline{\sigma}(x), \{\overline{\alpha}_\gamma(x) : \gamma \in \Gamma\}\rangle$$

of polymorphic formulas and extended terms over a copy set $C = \{1, ..., m\}$. It assigns to each Σ-structure A a Γ-structure B whose output domain is defined by the polymorphic formula $\overline{\varphi}(x)$ as $|B| = \{a^c : A \models \varphi_c[a], a \in |A|, c \in C\}$, drawn from m disjoint copies of the input domain $|A| \times C$. Adjacency between these elements is determined by the polymorphic extended terms $\overline{\pi}(x)$ and $\overline{\sigma}(x)$ representing the predecessor and successor functions, respectively. The collection of polymorphic formulas $\{\overline{\alpha}_\gamma(x) : \gamma \in \Gamma\}$ determine a partition of $|B|$ into $|\Gamma|$ pieces $\{a^c \in |B| : A \models \overline{\alpha}_\gamma[a^c]\}$ where the symbol γ appears.

Because the formulas above depend on only one variable, we call these *monadic* interpretations. Moreover, we will study quantifier-free interpretations exclusively, as we have been careful to define extended terms so that conditional definitions can be incorporated, as we saw in the example.[5]

22.3 Intuition of the proof

Before presenting the actual proof that ISL functions are quantifier-free logical interpretations, in this section we first walk through a few examples of actual phonological processes represented as quantifier-free interpretations. Specifically we will demonstrate a quantifier-free interpretation for an example of 1) substitution, 2) epenthesis, and 3) deletion.

[5]See the appendix for how we can effectively determine whether the successor and predecessor functions of an interpretation consistently define adjacency.

22.3.1 Substitution

For an example of substitution, we return to the 2-ISL postnasal voicing process from the introduction. The FST for this process was given in Figure 22.1; we repeat it here in Figure 22.4.

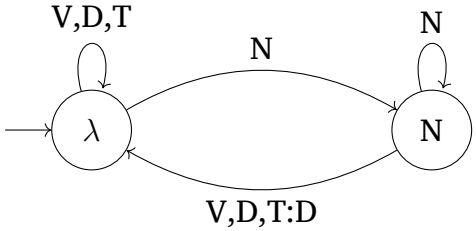

Figure 22.4: A second-degree ($k = 2$) ISL FST for the postnasal obstruent voicing map. Recall that this FST is minimized: states V, D, and T are collapsed with the start state.

We use the (hypothetical) input string /pampa/, the graph representation of which is shown in Figure 22.5.

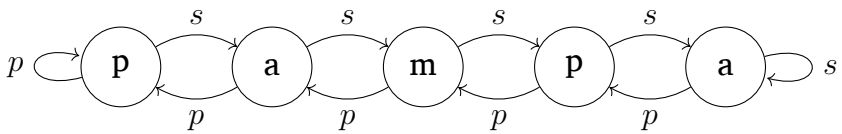

Figure 22.5: The structure for the input *pampa*, where $U_{\text{p}} = \{1, 4\}, U_{\text{a}} = \{2, 5\}$, and $U_{\text{m}} = \{3\}$

As the longest output string in the FST is of length 1, the copy set is $C = \{1\}$, and so the output graph is constructed from positions $1^1, 2^1, 3^1, 4^1$, and 5^1. We determine the labels of these output positions using the state we are 'in' for each position of the input graph. Determining which state we are in requires inspecting the neighboring positions. How many neighboring positions we inspect depends on 1) the degree k of the function, and 2) how close we are to the beginning of the string.

For the second point, there are three scenarios we are concerned with: 1) we are at the first position, 2) we are at some position j that is between the first and the k^{th} positions, or 3) we are at the k^{th} position or higher. Which of these scenarios we are in can be determined using the predecessor

function (details given in the next section), and will be designated with the terms $\psi_1(x), \psi_j(x)$, and $\psi_k(x)$.

For example, since at position 1 it is the case that $p(1) = 1$, we are in the first position and so $\psi_1(x)$ is true. For the remaining positions, $\psi_2(x)$ is true. (Since in this example $k = 2$, there are no positions between 1 and k and so there are only two possible scenarios.)

Which $\psi_i(x)$ is true tells us how many predecessors to look at from our current position. In the first position, since $\psi_1(x)$ is true we look at $1 - 1 = 0$ previous positions. For the remaining positions, $\psi_2(x)$ is true and so we look at $2 - 1 = 1$ previous position. We then make use of a set of formulas, one for each state, that assert which state we must be in using this information.

For example, positions 3 and 4 of Figure 22.5 satisfy the state formulas in 17a and 17b, respectively:

(17) a. $\theta_a(x) \equiv U_a(p(x)) \wedge \psi_2(x)$ [meaning the previous symbol was a.]

b. $\theta_m(x) \equiv U_m(p(x)) \wedge \psi_2(x)$ [meaning the previous symbol was m.]

So at position 3 we are in state a and at position 4 we are in state m. Note again though that the FST in Figure 22.4 uses an abbreviated alphabet for readability, such that there is a discrepancy in our 'state' formulas and the state labels of the diagram. Instead of a single state 'N', the complete FST would have a state for each nasal consonant in the language (i.e., {m, n, ɲ}). Formulas like $\theta_m(x)$ refer to this complete version of the FST.

Now using the transition function of the FST, we can determine the labels of each position of the output. Again using position 3 as an example, the formula in 17a is true when $x = 3$, as is the formula $U_m(x)$, since position 3 is labeled m. The value of the transition function for these arguments is $\delta_1(a, m) = m$. Together this means the formula in 18 is true for output position 3^1, and so it will be labeled m.

(18) $\alpha_m(x^1) \equiv \theta_a(x) \wedge U_m(x)$ [If the previous symbol was a, output m on input m.]

Likewise, the change from p to b takes place because at position 4 in the input graph, $\theta_m(x)$ is true (i.e., we are in state m) and $U_p(x)$ is true (i.e., input position 4 is labeled p). Since in the FST, $\delta_1(m, p) = b$, position 4^1 in the output graph is labeled b.

(19) $\alpha_b(x^1) \equiv \theta_m(x) \wedge U_p(x)$ [If the previous symbol was m, output b on input p.]

Lastly, the successor and predecessor functions are defined the same as the input to establish the linear order of the output positions. The resulting output graph is shown in Figure 22.6.

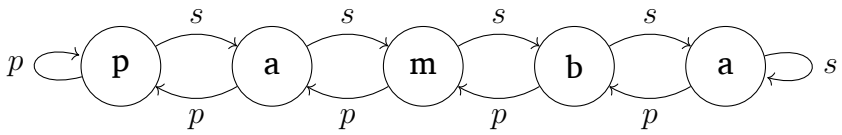

Figure 22.6: The output $pamba$, where $U_{\mathrm{p}} = \{1^1\}, U_{\mathrm{a}} = \{2^1, 5^1\}, U_{\mathrm{m}} = \{3^1\}$, and $U_{\mathrm{b}} = \{4^1\}$

Any substitution process, in which one segment is simply replaced with another, is a *same-length* function, one in which the output strings of the FST are always of length 1. In contrast, in epenthesis and deletion processes the length of the output strings can vary. In the next two subsections we present an example of each of these to demonstrate how that difference is handled through the copy set and the definition of predecessor and successor by cases.

22.3.2 Epenthesis

As an example of epenthesis we consider a process in Karo Batak (Woollams, 1996), in which a labiovelar glide (w) is inserted between two vowels when the first is back.[6] A rule for this process is given in 20, with examples in 21.

(20) $\varnothing \rightarrow$ w / [+syl, −front] __ [+syl]

(21) Karo Batak (Austronesian; Woollams, 1996)

a.	/ue/	[uwe]	‘yes’
b.	/doah/	[dowah]	‘carry a child in a sling’
c.	/dibərue/	[dibəruwe]	‘that woman’

[6] Woollams (1996, pg. 29) describes the process as optional. We do not address in this chapter how to handle optionality in this framework, but the use of probabilistic FSTs and/or weighted logics to model optionality could be a fruitful direction for future research.

The second-degree ($k=2$) ISL FST for this process is given in Figure 22.7. Again for readability the FST is minimized and has an abbreviated alphabet of {C, O, A}, where C=consonant, O=back vowel, and A=front or central vowel.[7] In addition to these, the output alphabet includes {w}.

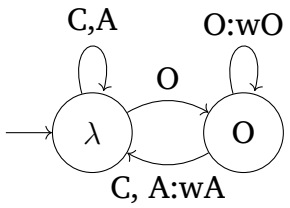

Figure 22.7: Second-degree ISL FST for intervocalic w-epenthesis map. This FST is minimized: states C and A are collapsed with the start state.

Figure 22.8 shows the structure for the input string /doah/.

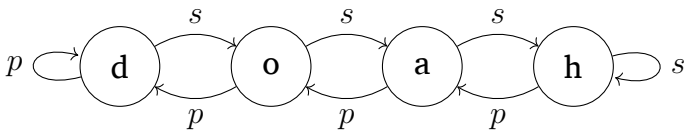

Figure 22.8: The input structure for *doah*, where $U_{\text{d}} = \{1\}, U_{\text{o}} = \{2\}, U_{\text{a}} = \{3\}$, and $U_{\text{h}} = \{4\}$

The addition of a segment in an epenthesis process is handled using the copy set. Recall that the copy set specifies how many copies of each input node are present in the output structure: for the substitution process presented in the previous section the copy set was $C = \{1\}$. For epenthesis (of a single segment), the copy set is $C = \{1, 2\}$. More generally, the copy set must be as large as the length of the longest output string.

This means that the domain of the output structure is potentially $\{1^1, 1^2, 2^1, 2^2, 3^1, 3^2, 4^1, 4^2\}$, but, as shown in Figure 22.9, not all of these positions are actually labeled, and not all of them are included in the domain and definition of the predecessor and successor functions. We describe first how these positions receive labels, and then how they are correctly ordered.

[7]The vowel inventory of Karo Batak includes two front vowels {i, e}, two central vowels {ə, a}, and three back vowels {ɯ, u, o}.

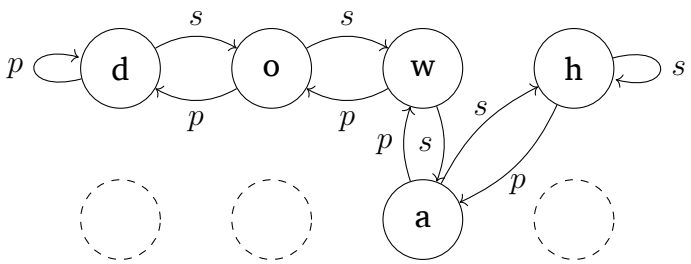

Figure 22.9: The output *dowah*, where $U_{\mathtt{d}} = \{1^1\}, U_{\mathtt{o}} = \{2^1\}, U_{\mathtt{w}} = \{3^1\}, U_{\mathtt{a}} = \{3^2\}$, and $U_{\mathtt{h}} = \{4^1\}$. Nodes drawn with dashed lines are omitted from the domain.

As in the previous example, an output position's label depends on which state we are in in the FST. Since again $k = 2$, which state we are in depends only on one preceding position. At position 3 (where the epenthesis will take place), the formula in 22 is true, meaning we are in state o.

(22) $\theta_o(x) \equiv U_o(p(x)) \wedge \psi_2(x)$ [meaning that the previous symbol was 'o']

Since position 3 is itself labeled a, the relevant transition in the FST is $\delta_1(o, a) = wa$. The length of this output string is 2, so the two formulas in 23 establish the labels of the output copies 3^1 and 3^2.

(23) a. $\alpha_w(x^1) \equiv \theta_o(x) \wedge U_a(x)$
b. $\alpha_a(x^2) \equiv \theta_o(x) \wedge U_a(x)$

What's left is to define the predecessor and successor functions to ensure that the output positions are ordered [dowah], as opposed to anything else (e.g., [doawh], [dohwa], [odhaw], etc.) This is done by cases, which we will go through it turn. (We will not go through the cases of the successor function, which parallel predecessor.) The predecessor arcs that fall under each case are color-coded in Figure 22.10.

For any copy that isn't the first, the predecessor is simply the prior copy. This is shown in red in Figure 22.10, for the copies of position 3.

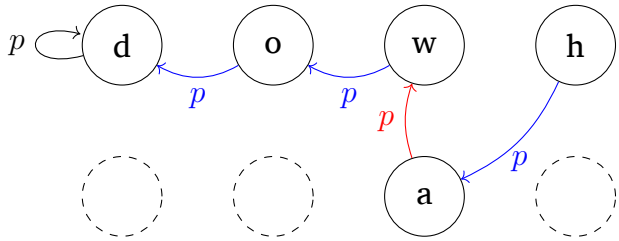

Figure 22.10: Defining the predecessor function

Otherwise, the predecessor is some copy of the preceding position, though importantly it needs to be the *last* copy of the preceding position. Since the number of copies can vary with position, we use a set of formula that indicate whether a given copy 'exists' (i.e., is labeled and should be included in the definition of predecessor/successor). These formulas are defined again in terms of the transition function, as whether a given copy exists depends on the length of the output string when a given position is read from a given state. A copy only exists if its number is less than or equal to the length of the output string at that point.

For example, we know copy 3^2 exists because $2 \leq |\delta_1(o, a)|$ and the formula in 24 is true when $x = 3$.

(24) $\varphi_2(x) \equiv \theta_o(x) \wedge U_a(x)$

But copies $1^2, 2^2$ and 4^2 do not exist, because the output strings of the FST for these positions are of length 1.

Getting back to the definition of predecessor, these $\varphi_c(x)$ formulas are used to verify that a prior position's copy exists *and* that it is the last one. For example, the predecessor of position 3^1 is 2^1, because 2^1 exists (i.e., $\varphi_1(2)$ is true) and 2^2 does not (i.e., $\varphi_2(2)$ is false). The predecessor arcs that fall under this case are shown in blue Figure 22.10.

Lastly, if none of the above cases are true, then the position must be the first one, and therefore it is its own predecessor.

This same method of ordering output copies extends straightforwardly to cases of deletion, an example of which we present in the next subsection.

22.3.3 Deletion

Our final example is an apocope process found in Mecayapan Isthmus Nahuat (Law, 1958; Kenstowicz and Kisseberth, 1979). As exemplified

in 26, unstressed vowels are (optionally) deleted word-finally when they follow a sequence of a vowel plus a voiced sonorant.

(25) V $\rightarrow \varnothing$ / V [+son, +voice] __ #

(26) Mecayapan Isthmus Nahuat (Uto-Aztecan; Mexico; Law, 1958; Kenstowicz and Kisseberth, 1979)

a.	/ʃikalili/	[ʃikalil]	'put it in it'
b.	/kitaja/	[kitaj]	'he already sees it'
c.	/kikowa/	[kikow]	'he buys it'
d.	/tami/	[tam]	'it ends'

The minimized FST for this process is shown in Figure 22.11. The alphabet is $\Sigma =$ {V, L, D}, where V $=$ vowel, L $=$ sonorant consonant, and D $=$ nonsonorant consonant.

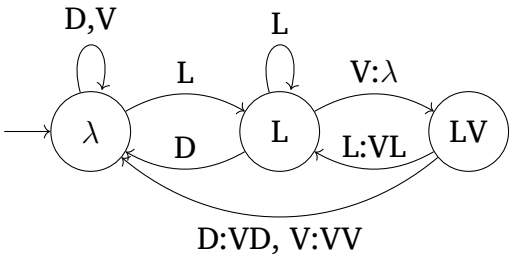

Figure 22.11: A third-degree ($k = 3$) ISL FST for word-final vowel deletion map.

We demonstrate the transduction using the example input /tami/, whose structure is below.

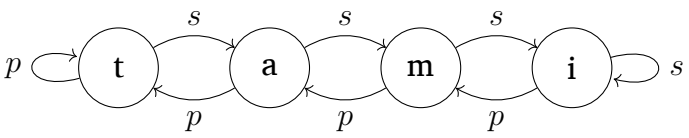

Figure 22.12: The input structure for *tami*, where $U_{\mathrm{t}} = \{1\}, U_{\mathrm{a}} = \{2\}, U_{\mathrm{m}} = \{3\}$, and $U_{\mathrm{i}} = \{4\}$.

Again we start with formulas that indicate which state we are in for each position. Since this process is third-order, these formulas reference

up to two prior positions. The formulas that evaluate to true for positions 1, 2, 3, and 4 of Figure 22.12 are given in 27a-27d, respectively. Recall that $\psi_1(x), \psi_2(x), \psi_3(x)$ indicate that we examine 0, 1, and 2 prior positions, respectively.

(27) a. $\theta_\lambda(x) \equiv \psi_1(x)$
b. $\theta_t(x) \equiv U_t(p(x)) \wedge \psi_2(x)$
c. $\theta_{ta}(x) \equiv U_t(p^2(x)) \wedge U_a(p(x)) \wedge \psi_3(x)$
d. $\theta_{am}(x) \equiv U_a(p^2(x)) \wedge U_m(p(x)) \wedge \psi_3(x)$

Up until the final vowel, all output positions are labeled straightforwardly using the FST transition function as in the previous examples. For example, copy 2^1 is labeled a because of the transition $\delta_1(t, a) = a$: the formula in 28 is true when $x = 2$ and $c = 1$.

(28) $\alpha_a(x^c) \equiv \theta_t(x) \wedge U_a(x)$

The 'deletion' takes place because copy 4^1, which corresponds to the input final vowel, does not receive a label and is not included in the output predecessor and successor functions (i.e., $\neg\varphi_1(x)$ is true when $x = 4$ and so the copy does not exist).

The resulting output structure is shown in Figure 22.13.

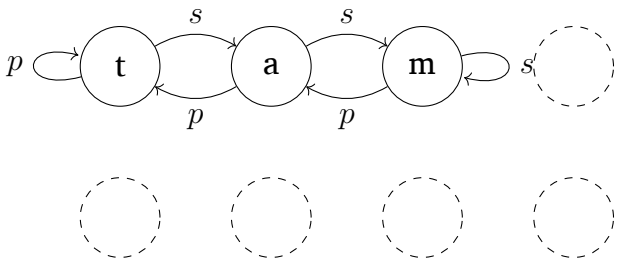

Figure 22.13: The structure for the string *tam*, where $U_{\mathrm{t}} = \{1^1\}, U_{\mathrm{a}} = \{2^1\}$, and $U_{\mathrm{m}} = \{3^1\}$

In the previous subsection we saw how the copy set and the definition of predecessor/successor by cases handles output strings of length > 1, and now we've seen a case where that length < 1. This example of deletion though in fact combines both of these scenarios, when a vowel follows a sonorant consonant but is not word-final. The FST handles this by

outputting λ for any vowel that follows a sonorant, but if it turns out to not be word-final (i.e., if anything else follows it), the vowel is 'returned' to the output on the subsequent transition.

In our logical translation of the FST, this means the returned vowel appears as one of the copies of the following position. For example, Figure 22.14 shows the output structure for the hypothetical input /lami/, in which both vowels follow a sonorant but only the second is word-final.

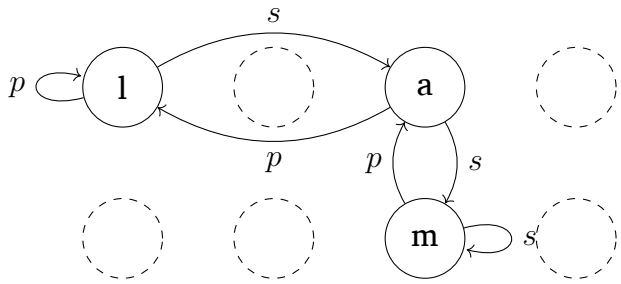

Figure 22.14: The structure for the string *lam*, where $U_{\text{l}} = \{1^1\}, U_{\text{a}} = \{3^1\}$, and $U_{\text{m}} = \{3^2\}$

Collectively, these examples of substitution, epenthesis, and deletion show how the local nature of the computation allows us to construct the output graph with quantifier-free formulas. The needed information for determining the output at any point is a bounded number of positions (i.e., a bounded number of calls to predecessor) away.

In the next section we formalize these results with a theorem of the equivalence between ISL FSTs (without null cycles) and quantifier-free logical interpretations.

22.4 Main Result

Chandlee (2014) and Chandlee *et al.* (2014) prove the equivalence of ISL FSTs (like those used in the previous section) and the class of ISL functions (Definition 8). Here we prove the equivalence of quantifier-free logical interpretations and ISL FSTs *without null cycles*. In other words, we prove that these interpretations describe exactly the class of ISL functions that are finite-to-one. This result is stated in Theorem 7.

We present the proof in two parts. First, we show that every ISL FST without null cycles can be translated into a quantifier-free logical interpretation over strings. Second, we show that a quantifier-free interpretation over strings can be computed by an ISL FST.

To state the theorem, we first define an interpretation as monotone if copies of an element maintain the same order as their originals. An example of a non-monotone interpretation is string reversal (e.g., any string w is mapped to the string with all symbols in w in reverse order). Such a map is phonologically implausible and falls outside of the ISL class (i.e., there is no k such that a k-ISL function can compute string reversal for strings of any length). In the following definition of monotone, we write $x+1$ and $x-1$ for the successor and predecessor of x respectively, and write $x \leq y$ to mean x precedes y (or equivalently y succeeds x).

Definition 14. An interpretation is *monotone* if $(x-1)^{c'} \leq x^{c} \leq (x+1)^{c'}$ for any copies c and c'. That is, any copy of $x \pm 1$ occurs after/before any copy of x. This means all copies of a given element remain together in blocks.

We now state the main result as Theorem 7.

Theorem 7. *A function $f\colon \Sigma^* \to \Gamma^*$ is input strictly local and finite-to-one if and only if it can be described by a monotone quantifier-free monadic interpretation.*

22.4.1 Every finite-to-one ISL function can be described by quantifier-free formulas

First we show that every finite-to-one ISL function can be described by quantifier-free formulas. We work from the function's FST, using the state labels to guide our formulas.

Let M be a k^{th}-degree local automaton without null cycles which computes f, with standard transitions between states recording the past $k-1$ symbols. For each state $q \in \Sigma^{<k} = Q$ and $a \in \Sigma$, let $\delta_1(q,a) \in \Gamma^*$ be the output string associated with each transition, and let $\rho(q)$ be the final output associated with each state.

State formulas

We first define formulas to describe what state we are in, based on the position in the input string, as represented by the variable x. Using p^i to

denote the i^{th} iterate of p (where p^0 is the identity), we can define formulas that monitor how close we are to the beginning:

$\psi_1(x) \equiv p(x) = x$ (we are at the first position)

$\psi_j(x) \equiv p^j(x) = p^{j-1}(x) \neq p^{j-2}(x)$ (we are at position j, for $1 < j < k$)

$\psi_k(x) \equiv p^{k-1}(x) \neq p^{k-2}(x)$ (we are at position k or higher)

The formula $\theta_q(x) \equiv U_{a_1}(p^i(x)) \wedge ... \wedge U_{a_i}(p(x)) \wedge \psi_{i+1}(x)$ says the machine is in state $q = a_1...a_i$ at position x of the input, for any $i = 0, ..., k-1$.

Size of the output structure

We create the output domain by *interleaving* copies of the input domain. The number of copies m required is given by the maximum length output string for any single input symbol, plus the maximum size of the final output (set $m = \max\{|\delta_1(q,a)| + |\rho(q)| : q \in Q, a \in \Sigma\}$).

The construction is complicated by the fact that the number of output symbols per input symbol can vary over the course of the computation. But using the state formulas, we can ascertain exactly how many copies of each input element are required by looking at the length of the output (anywhere from 0 to m) at that point in the automaton. The only complication is there might be final output $\rho(q)$ from any state q when we at the last element of the input, when $s(x) = x$. This increases the number of copies by the length of that output.

Let the output domain be defined as:

$$\varphi_c(x) \equiv \quad s(x) \neq x \wedge \bigvee\{\theta_q(x) \wedge U_a(x) : \text{when } q \in Q \text{ and } a \in \Sigma, \text{ for } c = 1, ..., |\delta_1(q,a)|\}$$

$$s(x) = x \wedge \bigvee\{\theta_q(x) \wedge U_a(x) : \text{when } q \in Q \text{ and } a \in \Sigma, \text{ for } c = 1, ..., |\delta_1(q,a)| + |\rho(q)|\}$$

That is to say, there will be one copy for each symbol in the output string $\delta_1(q,a)$ whenever we are in state q reading symbol a (none if the output string is empty). The second case is required to make additional copies for the final output.

Ordering the copies

We interleave the copies in such a way as to keep all outputs of a given input element x consecutive in copy order x^1, x^2, ..., ensuring monotonicity.

Here is the definition by cases for just the predecessor function, because the successor is similar. For clarity, let us say that x^c exists when $\varphi_c(x)$ holds and that it doesn't when $\neg\varphi_c(x)$. Also, we will use the more common notation $(x - i)$ instead of $p^i(x)$. Also, note the predecessor is by definition only applied to extant copies. So defining $p(x^c) = (x^c - 1)$ implies that x^c exists in the following.

$$x^c - 1 = \begin{cases} x^{c-1} & \text{for } c > 1, \text{ else } (c = 1) \\ x^1 & \text{if } x - 1 = x \\ (x-1)^{c'} & \text{if } (x-1)^{c'} \text{ exists and } (x-1)^{c'+1} \text{ doesn't, else} \\ \vdots & \vdots \\ (x-|Q|)^{c'} & \text{if } (x-|Q|)^{c'} \text{ exists and } (x-|Q|)^{c'+1} \text{ doesn't} \end{cases}$$

The first line says that the $(c-1)^{st}$ copy of an element is the predecessor of the c^{th} copy, whenever $c > 1$ (recall that if the c^{th} copy exists, so do all smaller copies). Otherwise, we need to find the last input position that has output. If there is no previous element because we are at the beginning, then just return oneself. If the previous element doesn't have any output (no copies of it exist) then we continue looking back. There is only a fixed distance backwards to look, because if there were more than $|Q|$ successive input positions without output, then the machine would contain a null cycle (by the pigeonhole principle).

To summarize, the predecessor of an output element is the preceding extant copy of that element, or else the last copy of the previous extant element. Failing that, we must be at the first copy of the first element, whose predecessor is itself.

Labeling the copies

Finally, we can define which symbols appear at each position of the output. Let

$$\alpha_\gamma(x^c) \equiv \quad s(x) \neq x \wedge \bigvee\{\theta_q(x) \wedge U_a(x) : \text{when } q \in Q \text{ and } a \in \Sigma, \text{ for } c = 1, ..., |\delta_1(q,a)|\}$$

$$s(x) = x \wedge \bigvee\{\theta_q(x) \wedge U_a(x) : \text{when } q \in Q \text{ and } a \in \Sigma, \text{ for } c = 1, ..., |\delta_1(q,a)\rho(q)|\}$$

which means that $\gamma \in \Gamma$ is placed on copy x^c of the output string.

22.4.2 Every quantifier-free interpretation is computable by an ISL FST

Preliminary steps

Although quantifier-free formulas can only see a local neighborhood, unlike local machines they can look both behind and ahead of the current location. But before we deal with this problem, we first straighten interpretations by rearranging copies so the output is generated in canonical order.

Recall that copies of the input domain used to define the output domain must preserve adjacency. In the output any copy of $x \pm 1$ must succeed/precede any copy of x. This means the blocks of copies retain the same order as the input. However, that leaves room for individual copies to be out of canonical order within each block. We rectify this by defining formulas $\beta_i(x^c)$ which detect if copy c of x is in position i within its own block, for $1 \leq i, c \leq m$, using the polymorphic definition of the predecessor function $\overline{\pi}(x)$ given by the interpretation. In the formula below, the notation (... $= x^d$) means $\varphi_d(...)$, and π^i refers to the i^{th} iterate of π.

$$\beta_i(x^c) \equiv \bigvee\{\pi^{i-1}(x^c) = x^d : 1 \leq d \leq m\} \wedge$$

$$[\bigwedge\{\pi^i(x^c) \neq x^d : 1 \leq d \leq m\} \vee \pi^i(x^c) = \pi^{i-1}(x^c) \neq \pi^{i-2}(x^c)]$$

where we leave off the very last inequality in case $i = 1$. It says the output position $x^c - (i-1)$ stays within its own block and that $x^c - i$ does not, unless of course we are in the first block, in which case $x^c - (i-1)$ is just at the bottom and $x^c - (i-2)$ is not (this last condition is left off when $i = 1$).

With this information in hand, we can rewrite the interpretation, so all copies are now in canonical order $i = 1, ..., m$. The original $\overline{\sigma}$ and $\overline{\pi}$ are replaced by the canonical $\overline{\sigma}'$ and $\overline{\pi}'$, like the definition by cases given in the previous direction of the proof. To rearrange the output formulas,

suppose $\alpha_\gamma(x^c)$ is the formula which tells us when copy c of x in the original interpretation was assigned to output symbol γ. Then define:

$$\alpha'_\gamma(x^i) \equiv \bigvee\{\beta_i(x^c) \wedge \alpha_\gamma(x^c) : 1 \leq c \leq m\}$$

which says we now output γ at position i in the canonical order just in case we originally output γ on copy c of x which was in position i of the original order. Observe that the domain information is already implicitly contained in the formulas $\beta_i(x^c)$, because copy c of x exists if and only if it is in some position i of its block. So, the new polymorphic domain formula is given by:

$$\varphi'(x^c) \equiv \bigvee\{\beta_i(x^c) : 1 \leq i \leq m\}$$

But since we have not actually changed the domain by re-ordering, this must be equivalent to the original φ. Now that we have put our interpretation in canonical order, we move on to the FST construction.

Construction of ISL FST

Definition 15. The *width* of a formula is its maximum embedding of predecessors or successors. The width of an interpretation is the maximum width of all of its formulas.

Intuitively, a width k formula is determined by the interval $[x-k, ..., x+k]$ around its free variable. More precisely, it does not depend on any information farther than k symbols away from that location.

Lemma 1. Let $\varphi(x)$ have width k, and let l refer to a position within a nonempty string $a_1...a_n$, $1 \leq l \leq n$. The prefix $u = a_1...a_{l-k-1}$ contains those symbols more than k positions before l (empty if $l \leq k+1$). The suffix $w = a_{l+k+1}...a_n$ contains those symbols more than k positions after l (empty if $l \geq n-k$). Let v be the substring that remains from removing the prefix u and suffix w, so that $a_1...a_n = uvw$. Then $uvw \models \varphi[l] \iff v \models \varphi[l - |u|]$.

Proof. A straightforward induction on the width k of $\varphi(x)$. □

In the lemma above, $1 \leq |v| \leq 2k+1$, as v consists of a_l along with at most k symbols on either side. We can think of v as $a_{l-k}...a_l...a_{l+k}$, where any subscript out of bounds is vacuous. When $k < l \leq n-k$ all those symbols

are in play and we can renumber them to let $v = b_1.....b_{k+1}....b_{2k+1}$ (see figure below). So the totality of $\varphi[1...n]$ can be determined by evaluating φ on substrings of length at most $2k+1$. Since a local machine of degree $2k+1$ has access to the previous $2k$ symbols along with the current one, we need to evaluate $\varphi[l]$ when the machine is at the end of v. We accomplish this by delaying the output for k steps, and producing the last k outputs as final output.

We construct a local transducer of degree $2k+1$ for which $Q = \Sigma^{<2k+1}$ and $\forall q \in Q, a \in \Sigma$, $\delta_2(q,a) = S^{2k}(qa)$, as defined in §22.2.1.[8] We determine the output transitions $\delta_1(q,b)$ for a given state $q = b_1...b_i$ and current symbol $b \in \Sigma$ as follows. The c^{th} symbol of $\delta_1(q,b)$ will be γ if $qb \models \overline{\alpha}_\gamma[(i{+}1{-}k)^c]$, for $1 \leq c \leq m$. Notice that this means there is no output until $|qb| = i+1 > k$, so $\delta_1(q,b)$ is empty for the first k steps when $0 \leq i < k$. At step k we begin producing output which corresponds to the first position in the input string. For the final output, we wrap up the last k values of the interpretation in one fell swoop, computing $q \models \overline{\alpha}_\gamma[i-(k-1)...i]$ by concatenating for $j = i-(k-1),...,i$ the outputs whose c^{th} symbol is $\gamma \iff q \models \overline{\alpha}_\gamma[j^c], 1 \leq c \leq m$. This relationship between the input and output is depicted in Figure 22.15.

$$\overbrace{a_1.....}^{u}\overbrace{a_{l-k}....a_l.......a_{l+k}}^{v}\overbrace{.......a_n}^{w} \models \phi[l]$$

$$\Leftrightarrow \quad b_1.....b_{k+1}....b_{2k+1} \models \phi[k+1]$$

$$\Leftrightarrow \quad qa \models \phi[|v|-k]$$

Figure 22.15: Illustration of the lemma, when $k < l \leq n-k$

Lastly, we show that the resulting machine does not a contain null cycle.

[8] Note there is a difference in how k is being used here compared to earlier in the chapter (and in the previous phonological literature). The value of k in the phonological examples presented throughout this chapter was determined by the length of the structural description of the process when described as a rule. In our FST construction, however, k is set to the maximum depth of calls to predecessor *or* successor, and the 'window' that can be examined is wide enough to cover k positions in *both* directions. In other words, if we converted the second-degree FST in Figure 22.4 into an interpretation, that interpretation would have a width of $k = 1$ (because its formulas contain at most one call to predecessor). If we then convert that interpretation back into an FST by the method described here, that FST would have degree 3 ($= 2k+1$).

If it did, there would be an input to reach and go around it arbitrarily many times. This would make the input distance between two adjacent output positions arbitrarily large, contradicting the finite width of an interpretation defining the successor and predecessor functions. Essentially, the bounded width of terms in a quantifier-free interpretation prohibits adjacency between output elements coming from input elements of unbounded distance apart.

22.5 Conclusion

We have seen that there is a machine-independent way of characterizing finite-to-one input strictly local functions. That characterization uses quantifier-free formulas in logic to monotonically translate between structures representing strings with adjacency. It is hoped that a similar logical characterization can be obtained for an analogous notion of output strictly local functions (Chandlee *et al.*, 2015) and strictly piecewise functions (Rogers *et al.*, 2010), which are conjectured to be a useful model for long-distance phonological maps such as vowel and consonant harmony.

The restriction placed on ISL functions—that they be finite-to-one—means the ISL class defined in this paper is in fact a proper subset of the one defined in Chandlee (2014). The restriction is a desirable one, however, as it restores closure under composition.[9] We omit the proof of this closure property because the details are complicated, but the intuition is as follows. A quantifier-free interpretation of a quantifier-free interpretation would involve only the substitution of terms—specifically the substitution of the input of one formula for the output of another. Thus no quantifiers are introduced, and so the resulting interpretation is also quantifier-free.

It is an empirical question whether there exist local phonological maps that do not hold this property of being finite-to-one (i.e., ones that map an infinite domain onto a finite co-domain), but under the assumption that no such phonological maps exists the class of functions defined in this chapter is in fact a better characterization of local phonology.

[9]Indeed the counterexample to closure under composition presented in Chandlee *et al.* (2018) includes a function with a null cycle.

22.6 Appendix: Deciding adjacency

Not every quantifier-free interpretation with the right kind of input and output signature necessarily determines a valid string to string mapping. The problem is that the defined predecessor and successor functions may not determine a valid adjacency relation. Indeed, they may not even be corresponding inverses of each other over the finite domain in question.

We can effectively check that a given interpretation determines a valid adjacency relation because monadic second-order logic is decidable over finite strings. To sketch the idea let $\pi(x)$ and $\sigma(x)$ be the purported predecessor and successor functions, respectively. A left endpoint is any element l that is not the successor of anything ($\forall x \sigma(x) \neq l$) and is its own predecessor ($\pi(l) = l$). Similarly, a right endpoint is any element r that is not the predecessor of anything ($\forall x \pi(x) \neq r$) and is its own successor ($\sigma(r) = r$).

First say there is exactly one left endpoint l, exactly one right endpoint r, and that predecessor and successor are mutually inverses of each other except at the endpoints: $x \neq l \rightarrow \sigma(\pi(x)) = x$, and $x \neq r \rightarrow \pi(\sigma(x)) = x$.

Every finite model satisfying these first-order conditions must consist of a single chain together with zero or more disjoint cycles. To remove the possibility of these additional cycles we need to say that either the structure is acyclic or connected (either one is sufficient). We can do this by universally quantifying over subsets of the domain that are closed under both predecessor and successor: $S(x) \rightarrow S(\pi(x)) \wedge S(\sigma(x))$.

These finite models are acyclic iff every nontrivial closed subset S contains an endpoint: $\forall S \neq \varnothing\ [S(x) \rightarrow S(\pi(x)) \wedge S(\sigma(x))] \rightarrow S(l) \vee S(r)$. Alternatively, we could say they are connected by stating every closed set containing an endpoint must be the entire structure. In either case, we have demonstrated that it is possible to write a sentence of monadic second-order logic which determines whether or not the purported predecessor and successor functions form the constituents of a valid adjacency relation.

