Quantifier-free Least Fixed Point Functions for Phonology

Jane Chandlee
Haverford College
Haverford, PA, USA
jchandlee@haverford.edu

Adam Jardine
Rutgers University
New Brunswick, NJ, USA
adam.jardine@rutgers.edu

Abstract

In this paper we define *quantifier-free least fixed point functions (QFLFP)* and argue that they are an appropriate and valuable approach to modeling phonological processes (construed as *input-output maps*). These functions, characterized in terms of first order logic interpretations over graphs, provide a close fit to the observed typology, capturing both local and long-distance phenomena, but are also restrictive in desirable ways. Namely, QFLFP logical functions approximate the computation of deterministic finite-state transducers, which have been argued to form a restrictive hypothesis for phonological processes.

1 Introduction

A lot of recent work in computational phonology has taken the approach of representing phonological processes as *maps* or *functions* from input strings/underlying representations to output strings/surface representations (Chandlee and Heinz, 2012; Heinz and Lai, 2013; Chandlee, 2014; Luo, 2017; Payne, 2017). The question of interest underlying such work is how computationally powerful or expressive the functions involved in phonology need to be. In particular, these authors have argued that various proper subsets of the regular relations (i.e., *subregular* functions) are sufficient to model the attested range of phonological phenomena. Special attention has been given to one such subclass, the *subsequential* functions, which are restrictive in being deterministic but still sufficiently expressive to capture the needed range of segmental phonological processes, both local and long-distance.

While this work has largely proceeded using the finite-state formalism, a related line of work has aimed to address similar questions using logical characterizations (Jardine, 2016; Chandlee and Jardine, 2019), one advantage of logic being that it offers a unified approach to modeling both linear (i.e., string-based) and non-linear (i.e., autosegmental representations, metrical structure) phenomena. However, a restrictive logic that is still sufficiently expressive to cover a wide range of phonological phenomena has yet to be established. Recent work has studied the relationships between classes of automata and logical transductions (Engelfriet and Hoogeboom, 2001; Filiot, 2015; Filiot and Reynier, 2016), but no work has studied a logical characterization of the subsequential functions.

This paper thus aims to fill this gap by proposing a type of logic that approximates the subsequential class of functions in an important way. This logic is quantifier-free least fixed point (QFLFP), which is first order logic *without* quantifiers but *with* a (monadic) least fixed point operator. We will define this type of logic and demonstrate how it can be used to model a range of phonological processes in a recursive and output-oriented way. We will also show that the functions can be defined with this type of logic are in fact a proper subset of the subsequential functions. This is interesting both because it shows how logical transductions can be related to subsequential functions, and also because it closely fits the attested typology of phonological processes.

The paper is organized as follows. In §2 we provide the needed preliminaries, and in §3 we give an overview of the hierarchy of function classes in the finite-state formalism. In §4 we provide relevant background on logical characterizations of functions, quantifier-free logic, and least fixed point operators, before defining QFLFP functions. In §5 we demonstrate the application of QFLFP functions to phonology by giving example analyses of phonological processes. In §6 we establish certain properties of QFLFP, and in §7 we discuss the implications of our findings and highlight

a few areas for future work. §8 concludes.

2 Preliminaries

Given a finite alphabet Σ, with Σ^* we designate the set of all possible finite strings of symbols from Σ and with $\Sigma^{\leq n}$ we designate the set of all possible finite strings of length $\leq n$. The length of a string w is $|w|$. The unique empty string is λ, so $|\lambda| = 0$.

Let $\mathcal{P}(X)$ denote the powerset of a set X.

For a set of strings L, $\texttt{pref}(L) = \{u \mid \exists v \text{ such that } uv \in L\}$, $\texttt{suff}(L) = \{v \mid \exists u \text{ such that } uv \in L\}$, and for a string w, $\texttt{pref}(w) = \{u \mid w = uv \text{ for some string } v\}$,

$$\texttt{pref}_n(w) = \begin{array}{ll} w & \text{if } |w| \leq n; \text{ otherwise} \\ w_1, & \text{where } w = w_1 w_2, |w_1| = n, \end{array}$$

$$\texttt{suff}_n(w) = \begin{array}{ll} w & \text{if } |w| \leq n; \text{ otherwise} \\ w_2, & \text{where } w = w_1 w_2, |w_2| = n. \end{array}$$

For a string $w = \sigma_1 ... \sigma_i ... \sigma_n$, let $w(i)$ denote σ_i.

For a set of strings $L \subseteq \Sigma^*$ and two strings $w, v \in \Sigma^*$, we write $w \equiv_L v$ iff $\forall z \in \Sigma^*$, $wz \in L \leftrightarrow vz \in L$. This is an equivalence relation on Σ^* inducing a partition P_L on Σ^*. A set L is *regular* if and only if P_L is finite. Note that if $w, v \in A$ for some $A \in P_L$, then for any $z \in \Sigma^*$ then $wz \in A' \leftrightarrow vz \in A'$ for some other $A' \in P$.

We can define a similar notion for functions on Σ^*. For a set of strings L, the *longest common prefix* is the longest shared prefix of all the strings in L. Formally,

$$\texttt{lcp}(L) = \begin{array}{l} u \in \bigcap_{w \in L} \texttt{pref}(w), |u| \geq |u'| \\ \text{for all } u' \in \bigcap_{w \in L} \texttt{pref}(w). \end{array}$$

For a function $f : \Sigma^* \to \Gamma^*$, and a string $w \in \Sigma^*$, the *tails* of w with respect to f are defined as follows:

$$\texttt{tails}_f(w) = \{(z, z') \mid \begin{array}{l} f(wz) = uz', \\ u = \texttt{lcp}(f(w\Sigma^*))\}. \end{array}$$

Two strings $w, v \in \Sigma^*$ are *tail-equivalent*, written $w \equiv_f v$ iff $\texttt{tails}_f(w) = \texttt{tails}_f(v)$. Note that $\equiv_f$ is an equivalence relation on Σ^*; let P_f be the partition it induces on Σ^*. A function f is *subsequential* (SUBSEQ) iff P_f is finite (Oncina et al., 1993).

We will make use of the following operation that takes the pairwise intersection of the blocks of two partitions P_1 and P_2 on some set X.

$$P_1 \otimes P_2 \stackrel{\text{def}}{=} \left\{ A \cap B \neq \varnothing \;\middle|\; A \in P_1, B \in P_2 \right\}$$

It is elementary to show that $P_1 \otimes P_2$ is also a partition on X, and that the operation is associative and commutative.

3 Transducers and function classes

It will be helpful to first define some important function classes and illustrate them with automata. The SUBSEQ class is exactly described by *subsequetial finite-state transducers* (SFSTs), or determinstic finite state machines that output a string for each input symbol and upon ending on a state (Schützenberger, 1977; Mohri, 1997). Note that this makes SUBSEQ a strict sublcass of the *regular* (or *rational*) functions, or exactly those describable with an FST (deterministic or otherwise). Formally, a SFST is a tuple $\langle Q, q_0, Q_f, \Sigma, \Gamma, \delta, \omega \rangle$ where Q is the set of states, $q_0 \in Q$ is the single initial state, $Q_f \subseteq Q$ is the set of final states, Σ and Γ are the input and output alphabets, respectively, $\delta : (Q \times \Sigma) \to (\Gamma^* \times Q)$ is a *transition function*, and $\omega : Q_f \to \Gamma^*$ is the *output function*. Note that because δ takes pairs in $(Q \times \Sigma)$ (that is, it does not have transitions on an input λ), and because it is a function, the machine is deterministic. We define the transitive closure δ^* of δ recursively in the usual way; i.e. $\delta^*(q, \lambda) = (\lambda, q)$ and if $\delta^*(q, w) = (v_1, q_1)$ and $\delta(q_1, \sigma) = (v_2, q_2)$ then $\delta^*(q, w\sigma) = (v_1 v_2, q_2)$.

A transducer $T = \langle Q, q_0, Q_f, \Sigma, \Gamma, \delta, \omega \rangle$ describes a function $f_T : \Sigma^* \to \Gamma^*$ defined as $f_T(w) = uv$ where $\delta^*(q_0, w) = (u, q_f)$ for some $q_f \in Q_f$, and $v = \omega(q_f)$. As an example, the SFST in Fig 1 models the function described by the rewrite rule in (1) ('change an a that follows a b to b'). (In all SFSTs in this paper, the initial state is indicated with a small unlabeled incoming arrow, all states are assumed to be final, and the output function is represented in the state label with the string to the right of the colon.)

(1) $a \to b \,/\, b \_\_$ (simultaneous)

As stated above, SUBSEQ is exactly the class of functions with a finite set of tail-equivalence classes. In the minimal SFST for a subsequential function, each state corresponds to a tail-equivalence class.

Restrictions on the nature of these tail-equivalence classes offer *subclasses* of SUBSEQ that have been argued to be relevant to phonology. One such class is the *input strictly local* (ISL) functions, a subsequential

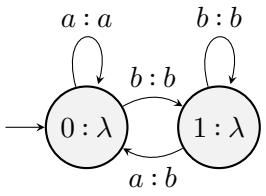

Figure 1: A SFST implementing the rule a → b / b __, applied simultaneously, for $\Sigma = \Gamma = \{a, b\}$. This function is ISL_2.

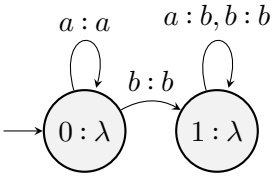

Figure 2: A SFST implementing the rule a → b / b __, applied iteratively, for $\Sigma = \Gamma = \{a, b\}$. This function is OSL_2.

class defined by Chandlee (2014), which Chandlee and Heinz (2018) argue can model any local phonological rule that applies simultaneously. Formally, the ISL class is defined as below.

Definition 1 (ISL; Chandlee 2014) *A function f is ISL_k iff for all strings w, v, $\texttt{suff}_{k-1}(w) = \texttt{suff}_{k-1}(v)$ implies $\texttt{tails}_f(w) = \texttt{tails}_f(v)$.*

Intuitively, the ISL class is exactly those functions for which, for any symbol in the string, its output is entirely decided by the preceding $k - 1$ symbols in the input. This means that any ISL_k function can be described by a SFST whose states correspond to the $k - 1$ suffixes of Σ^*. Fig. 1 is exactly one such SFST: its states represent the previous 1 symbol in the input—state 0 represents a preceding a in the input, and state 1 a preceding b. The function is thus ISL_2.

Another notion of local string functions is those for which the output of a string is based on the preceding $k - 1$ symbols in the *output*; this is the *output strictly local* (OSL) class. A definition for this class is given below.

Definition 2 (OSL; Chandlee 2014) *A function f is OSL_k iff for all strings w, v, $\texttt{suff}_{k-1}(f(w)) = \texttt{suff}_{k-1}(f(v))$ implies $\texttt{tails}_f(w) = \texttt{tails}_f(v)$.*

(Note: this is an incomplete definition, but for the purposes of this paper it is sufficient. For the complete definition see Chandlee et al. 2015.) The definition for OSL is parallel to that for ISL, except it refers to the output of the function. An example is given in Fig. 2 for the rule in (2), applied iteratively.

(2) $a \rightarrow b$ / b __ (iterative)

By "applied iteratively," we mean that an a becomes a b after a b which may have been present in the input (and so remained a b in the output) or may have been an a in the input (and so was itself turned into a b in the output). This is OSL_2, because (either way) whether or not an input a is output as b depends on the immediately preceding *output* symbol.

The way in which ISL and OSL correspond to the difference between simultaneous and iterative application of rules can be seen more clearly when one compares the output strings for these two FSTs for the input $baabaa$. The FST in Fig. 1 outputs $bbabba$ for this input string, such that all a's that follow b's *in the input* are changed to b. In contrast, the FST in Fig. 2 outputs $bbbbbb$, such that all a's are changed to b. The structural difference in the FSTs that is responsible for this difference is that in Fig. 1 all transitions with the same input symbol lead to the same state whereas in Fig. 2 all transitions with the same output string lead to the same state.[1]

The ISL and OSL functions are strict subclasses of SUBSEQ. That ISL $\subsetneq$ SUBSEQ is witnessed by Fig. 3. The SFST in this figure represents a function that changes a to b if it follows a b, where any number of c's can intervene. The strings c^k and bc^k clearly have the same suffix of length $k - 1$ (for any k), but (contra Definition 1) they have different tails, as (a, a) is in the tails of c^k while (a, b) is in the tails of bc^k.

That OSL $\subsetneq$ SUBSEQ is also witnessed by Fig. 3. The strings $f(c^k) = c^k$ and $f(bc^k) = bc^k$ again have the same suffix of length $k - 1$ (for any k), but (contra Definition 2) they have different tails, as (a, a) is in the tails of c^k while (a, b) is in the tails of bc^k.

The ISL and OSL classes are thus restrictive classes of functions that have been posited as strong hypotheses for the computational complexity of phonological processes with local triggers (Chandlee, 2014; Chandlee and Heinz, 2018). Examples of their connection to phonological pro-

[1] For more on how FSTs model modes of rule application see Kaplan and Kay (1994) and Hulden (2009).

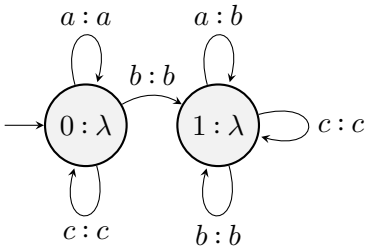

Figure 3: A SFST changing any a following a b in a word, regardless of any number of intervening c's, to a b. This function is neither ISL or OSL.

cesses are given below. The purpose of this paper is to investigate, through logical transductions, a class that generalizes the intuitions behind these classes, and also captures non-ISL and non-OSL functions, like Fig. 3, that are relevant to phonology.

4 Logical transductions

We consider finite strings models over the signature $\mathcal{I}$ of strings in Σ with predecessor function p, delimited by boundaries $\rtimes$ and $\ltimes$, and assume that for a string of length n (including the boundaries) the domain of its model is $D = \{1, ..., n\}$.

$$\mathcal{I} = \{p, \rtimes, \ltimes, P_{\sigma \in \Sigma}\}$$

Figure 4 gives a model for a string over the alphabet $\Sigma = \{a, b\}$ with $D = \{1, 2, 3, 4, 5, 6, 7\}$. For each $\sigma \in \Sigma \cup \{\rtimes, \ltimes\}$, $P_\sigma \subseteq D$ includes those members of D that are labeled with σ. For example, in the figure $P_b = \{3, 4, 6\}$.

$\rtimes_1$	a_2	b_3	b_4	a_5	b_6	$\ltimes_7$

Figure 4: Model for the string $abbab$

The signature also includes a predecessor function p that gives the immediately proceeding position for each position—e.g., $p(3) = 2, p(2) = 1$, etc.

We define a monadic second order (MSO) logic $\mathcal{L}_\mathcal{I}$ over $\mathcal{I}$ in the usual way; that is, $\mathcal{L}_\mathcal{I}$ is a predicate logic whose atomic formulae are of the form $\sigma(t)$, where $\sigma(t)$ is true when the *term* t is interpreted as a member of P_σ. More formally, a term is either some member x of a countably infinite set of first-order variables (which range over the domains of models) or the application $p(t)$ of the predecessor function to a term t. MSO is then a logic defined recursively in which $\sigma(t)$ for some t and $\sigma \in \Sigma$ is a *well-formed formula* (WFF); $X(t)$ for some term t and some variable X drawn from a countably infinite set of set variables is a WFF; for WFFs φ and ψ, $\neg\varphi$ and $\varphi \vee \psi$ are WFFs; and for WFFs $\varphi(x)$ and $\psi(X)$ with free first-order and set variables x and X, respectively, $(\exists x)[\varphi(x)]$ and $(\exists X)[\psi(X)]$ are WFFs. As shorthand, for $k \geq 0$, $p^k(x)$ denotes x when $k = 0$, $p(p^{k-1}(x))$ otherwise. The semantics of $\mathcal{L}_\mathcal{I}$ are defined as usual over string models with the first-order variables mapped to positions in a string and second-order variables mapped to sets of positions in the string. In particular we write $\mathcal{S} \vDash \varphi(x)[x \mapsto i]$ for a string model $\mathcal{S}$ that satisfies $\varphi(x)$ when x is mapped to $i \in D$.

For example, $b(p(x))$ is a WFF in $\mathcal{L}_\mathcal{I}$, as is $b(x) \wedge b(p(x))$. The string in Fig. 4 satisfies $b(p(x))$ when x is mapped to positions 4, 5, or 7 (that is, the positions whose predecessor is labeled b). The string in Fig. 4 satisfies $b(x) \wedge b(p(x))$ only when x is mapped to 4, as that is the only position that is both a b and is immediately preceded by a b.

We can define functions logically through a *logical interpretation* of an output signature $\mathcal{O}$ in the logic $\mathcal{L}_\mathcal{I}$ of the input signature. Specifically, a MSO transduction T to the output signature $\mathcal{O}$ over strings in Γ

$$\mathcal{O} = \{<', P_{\gamma \in \Gamma}\}$$

is a finite ordered *copy set* $C = \{1, ..., n\}$ and a series of formulae $\gamma^c(x)$ in $\mathcal{L}_\mathcal{I}$ with exactly one free variable x and for each $\gamma \in \Gamma$ and $c \in C$. The copy set allows us to create up to n output copies of each input element.[2] See below for an example demonstrating the use of multiple copies of an input element.

We define the semantics of T following Engelfriet and Hoogeboom (2001). For an input string model $\mathcal{S}$ over the signature $\mathcal{I}$ with domain D, its output $T(\mathcal{S})$ is a string $\mathcal{S}'$ over the signature $\mathcal{O}$ with domain D' and unary relations $P_{\gamma \in \Gamma}$ built in the following way. For each $d \in D$, there is a $d^c \in D'$ that belongs to P_γ if and only if $\mathcal{S} \vDash \gamma^c(x)[x \mapsto d]$ for exactly one $\gamma \in \Gamma$ and exactly one $c \in C$. If no such γ or c exists, then no output position is constructed for d. We assume

[2] We can also consider a closed domain formula φ_{dom} which specifies the domain of the function; as long as this formula is in MSO or lower, this does not change any of the below results, so we do not discuss it in detail here. For more see Engelfriet and Hoogeboom (2001); Filiot (2015).

that the transduction is *order-preserving*; that is, the output order $<'$ in $\mathcal{O}$ is such that for any $d \in D$ and $i < j \in \mathcal{C}$, $d^i <' d^j$, and for any distinct $d, e \in D$ and $i, j \in \mathcal{C}$, for the copies d^i and e^j in D', $d^i <' e^j$ if and only if $d < e$.

Thus, a MSO transduction T thus defines a function $f(T)$ from strings in $\rtimes\Sigma^*\ltimes$ to strings in Γ^*.

As an example, we logically define the function for 'change a b following another b to a c' rule given in (3), assuming $\Sigma = \{a, b\}$ and $\Gamma = \{a, b, c\}$.

(3) $b \rightarrow c \,/\, b\_\_$

Since this rule makes no mention of a's, all positions in the input model that are labeled a can likewise be labeled a in the output model. The formula in (4-a) achieves this. As this is a 'substitution' rule that doesn't extend the length of the string, only a single copy of each input position is needed. Therefore we simply mark the output relations with a prime γ' instead of a number.

(4) a. $a'(x) \stackrel{\text{def}}{=} a(x)$
b. $b'(x) \stackrel{\text{def}}{=} b(x) \wedge \neg b(p(x))$
c. $c'(x) \stackrel{\text{def}}{=} b(x) \wedge b(p(x))$

Likewise, the formula in (4-b) declares which positions in the output model are labeled b: namely, those positions that are labeled b in the input model but whose predecessors are *not* also labeled b (so excluding those b's that are subject to the rule in (3)). Lastly, the formula in (4-c) declares which positions are labeled c in the output model: those positions that are subject to the rule in (3). The collective result is the output model shown in Figure 5.

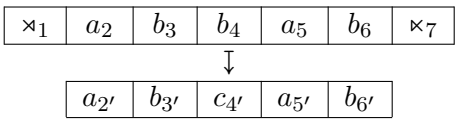

Figure 5: Transduction of *abbab* following (4).

The following example demonstrates the use of a copy set with a size larger than 1. With the following formulas and a copy set $C = \{1, 2\}$ we define a function that inserts a c after every input b.

(5) a. $a^1(x) \stackrel{\text{def}}{=} a(x)$
b. $a^2(x) \stackrel{\text{def}}{=} \texttt{False}$
c. $b^1(x) \stackrel{\text{def}}{=} b(x)$
d. $b^2(x) \stackrel{\text{def}}{=} \texttt{False}$
e. $c^1(x) \stackrel{\text{def}}{=} \texttt{False}$
f. $c^2(x) \stackrel{\text{def}}{=} b(x)$

The formulas (5-a) and (5-c) create a first copy labeled a and b in the output, respectively, for each a and b in the input. The formula (5-f), then, creates a second copy labeled c for each b in the input. As the other formula are set to `False`, no other copies are produced in the output. Finally, following the precedence relation defined above, each 1 copy b precedes the 2 copy c. Thus, *abbab* is output as *abcbcabc*.

Fig. 6 illustrates the transduction defined by these formulas with the input string *abbab*.

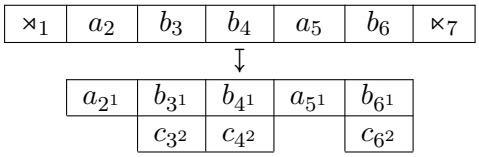

Figure 6: Transduction of *abbab* into *abcbcabc* following (5)

We fix the precedence relations in the output due to the following equivalence.

Theorem 1 (Filiot 2015) *A function f is regular iff there is an order-preserving monadic second order transduction T such that $f = f(T)$.*

Theorem 1 thus guarantees that any logic and signature we use that is less than or equal to monadic second order logic in expressivity will give us (sub-)regular functions.

4.1 Quantifier-free logic

Note that none of the formulas used in the previous example included quantifiers. Let *quantifier-free* (QF) denote the restriction on $\mathcal{L}_\mathcal{I}$ to formula with no quantifiers (and thus no set variables).

Chandlee and Lindell (in prep.) relate QF to the ISL class, showing that any ISL function with bounded deletion is QF-definable. However, their definitions do not consider functions with the output order as defined above. We show in §6.2 that, given this order, QF = ISL exactly.

An example of a phonological rule that cannot be modeled with this restricted logic is unbounded iterative spreading of a feature, such as the spreading of nasality from a nasal consonant to a following sequence of vowels: $nVVV \mapsto n\tilde{V}\tilde{V}\tilde{V}$. The

formula in (6) would achieve the nasalization of the first vowel following the input nasal.

(6) $\tilde{V}'(x) \stackrel{\text{def}}{=} n(p(x))$

The problem with modeling this process using only QF formula arises when trying to account for the nasalization of the second and third vowels. Of course we can add to (6) $n(p^2(x)) \vee n(p^3(x))$, which would accommodate the present example. But then what about a form like /nVVVV/, for which the nasalization of the final vowel would require a formula to identify the nasality of the predecessor of its predecessor of its predecessor of its predecessor (i.e., $p^4(x)$)? The issue now should be clear: for an unbounded pattern such as this one, the input nasal that triggers the nasalization of a given vowel may be an unbounded number of segments away. Without knowing in advance how large of a bound that is, a formula in terms of the predecessor function cannot be defined. Instead, what is needed is a quantifier: nasalize a vowel if there *exists* a nasal consonant at a prior position.

The traditional analysis of a feature spreading process like this one is that after the first nasalization of the vowel immediately following the nasal consonant, the actual trigger of the next nasalization is the most 'recently' nasalized vowel. But capturing that intuition requires specifying something about the labels in the *output* model, not the input model, which falls beyond the capabilities of QF. In that sense the limitation here is not the need for the quantifier, but the restriction on the formula to referring to the input model only.

In the next section we will show how to model such output-oriented processes using QF formula with the addition of the *least fixed point* (`lfp`) operator.

4.2 Least fixed point quantifier-free logic

Least fixed point logic allows us to add inductive definitions to our logics. The following is based on Libkin (2004, Ch. 10), but simplified for unary predicates.

For a set U an *operator* on U is a function $F : \mathcal{P}(U) \to \mathcal{P}(U)$. A set $X \subseteq U$ is a *fixed point* if $F(X) = X$. A set X is the *least fixed point* $\texttt{lfp}(F)$ of F iff it is a fixed point of F and for every other fixed point Y of F, $X \subseteq Y$. We write $\texttt{lfp}(F)$ for the least fixed point of F.

An operator F is *monotone* if $X \subseteq Y$ implies $F(X) \subseteq F(Y)$. For every monotone operator F, $\texttt{lfp}(F) = \bigcup_i X^i$, where each X^i is from the sequence in (7).

(7) $X^0 = \varnothing, X^{i+1} = F(X^i)$

That is, $\texttt{lfp}(F)$ is the set that is converged to by recursive applications of F. Given models with finite domains, there is always such a (finite) set for a monotone operator on the domain.

For a signature $\mathcal{S}$ and any model $\mathcal{M}$ in the signature whose universe is M, we extend our logic with an additional set predicate A not in $\mathcal{S}$. A formula $\varphi(A, x)$ with a single free variable x and free set variable A induces an operator F_φ on M as follows.

$$F_\varphi(X) = \left\{m \;\middle|\; \mathcal{M} \vDash \varphi(A, x)[A \mapsto X, x \mapsto m]\right\}$$

We will work through an example to illustrate, using the string model in Figure 7. Let $\varphi(A, x)$ be defined as in (8).

$\rtimes_1$	a_2	b_3	a_4	a_5	a_6	c_7	a_8	$\ltimes_9$

Figure 7: Model for the string $abaaaca$

(8) $\varphi(A, x) = a(x) \wedge (b(p(x)) \vee A(p(x)))$

First let $X^0 = \varnothing$. We then have

$$F_\varphi(\varnothing) = \{m \mid D \vDash \varphi(A, x)[A \mapsto \varnothing, x \mapsto m]\}.$$

No position $m \in D$ satisfies $A(p(x))$ since $A = \varnothing$, but position 4 satisfies both $a(x)$ and $b(p(x))$, and so $F_\varphi(\varnothing) = \{4\}$.

Now $X = \{4\}$, so $F_\varphi(\{4\}) = \{m \mid D \vDash \varphi(A, x)[A \mapsto \{4\}, x \mapsto m]\}$. Now position 5 satisfies the formula (because it is labeled a and its predecessor is in A), so $F_\varphi(\{4\}) = \{4, 5\}$.

Now with $X = \{4, 5\}$, $F_\varphi(\{4, 5\}) = \{4, 5, 6\}$, and then $F_\varphi(\{4, 5, 6\}) = \{4, 5, 6\}$, so $\{4, 5, 6\}$ is $\texttt{lfp}(F_\varphi)$, as no new elements are added.

Whether or not F_φ is monotone (and thus, whether $\texttt{lfp}(F_\varphi)$ can be determined via the sequence in (7)) is undecidable for an arbitrary FO formula φ. So the least fixed point extension of FO is defined with the following restriction on the set variable A:

(9) A is *positive* in φ; that is, it is under the scope of an even number of negations.

We then extend the usual definition of QF with the following syntax:

Definition 3 (QFLFP **syntax)** *Any formula in* QF *is in* QFLFP.

For a formula $\varphi(A, y)$ *in* QF *extended with the predicate* $A(y)$ *satisfying (9),* $[\texttt{lfp}\ \varphi(A, y)](x)$ *is a formula in* QFLFP.

A formula $[\texttt{lfp}\ \varphi(A, y)](x)$ is then true when x is interpreted as an element in the least-fixed point of the operator induced by $\varphi(A, y)$.

Definition 4 (QFLFP **semantics)** *For any formula in* QFLFP *that is also in* QF*, satisfaction is the same as in* QF.

For a formula $[\texttt{lfp}\ \varphi(A, y)](x)$ *in* QFLFP*,*

$$\mathcal{M} \vDash [\texttt{lfp}\ \varphi(A, y)](x)[x \mapsto m] \text{ iff } m \in \texttt{lfp}(F_\varphi)$$

In the next section we will demonstrate such transductions with phonological examples.

5 QFLFP functions in phonology

First we return to the example of unbounded iterative spreading, discussed in the previous section as a case that can't be modeled as a QF transduction. For simplicity we will assume the alphabet $\Sigma = \{a, b\}$ and use the model in Figure 8.

$\rtimes_1$	b_2	a_3	a_4	a_5	$\ltimes_6$

Figure 8: Model for the string *baaa*

The rule is the same as in (2), repeated below in (10).

(10) $a \to b \,/\, b\_\_$ (iterative)

The expected output string is then *bbbb*. The formula in (11) uses a `lfp` operator to declare which output positions should be labeled b.

(11) $b'(x) \stackrel{\text{def}}{=} [\texttt{lfp}\ b(y) \vee A(p(y))](x) \wedge \neg \ltimes (x)$

At first only position 2 satisfies $b(y) \vee A(p(y))$, since initially $A = \varnothing$ and so no positions satisfy $A(p(y))$. Now with $A = \{2\}$, position 3 satisfies $b(x) \vee A(p(y))$, since its predecessor is in A. Next 4 can be added, and then 5, and finally 6. After that no new positions will ever be added, and so $\{2, 3, 4, 5, 6\}$ is the `lfp`. The copies of theses positions are all labeled b in the output—with the exception of 6 due to $\neg \ltimes (x)$. Thus the map *baaa* $\mapsto$ *bbbb* is obtained.

We also consider a case of unbounded spreading with blocking, in which the feature spreads unboundedly but only up to a particular type of segment, a blocker. This type of process is attested in Johore Malay (Onn, 1980), where nasality spreads from a nasal consonant through a following span of vowels and glides but stops when it reaches an obstruent:

(12) Johore Malay (Onn, 1980)
/pəŋawasan/ $\mapsto$ [pəŋãwãsan], 'supervision'

We will extend our analysis of unbounded spreading to this case as well, using the alphabet $\Sigma = \{a, b, c\}$, where c is a blocker.

$\rtimes_1$	b_2	a_3	a_4	c_5	a_6	b_7	a_8	$\ltimes_9$

Figure 9: Model for the string *baacaba*

As (13) shows, the change is minor: the conditions under which a position is added to the set are made more stringent by including $\neg c(x)$.

(13) $b'(x) \stackrel{\text{def}}{=} [\texttt{lfp}(b(y) \vee (A(p(y)) \wedge \neg c(y)))](x) \wedge \neg \ltimes (x)$

The effect of this added restriction is that the set will be built as follows:

$\varnothing$
$\{2, 7\}$
$\{2, 3, 7, 8\}$
$\{2, 3, 4, 7, 8, 9\}$

This last set is the `lfp`, which picks out those positions that will be labeled b in the output (again except for 9), giving the string *bbbcabb*. The a that follows the blocker c will not receive the spreading feature because it is never added to the set, because its predecessor is prevented from joining the set by being labeled c.

Lastly, we consider a case of unbounded agreement, where a segment takes on a feature from a triggering segment but (unlike in spreading) the intervening segments are unaffected. An example is in Kikongo, in which a liquid becomes nasal (underlined below) following a nasal somewhere in a root (Ao, 1991; Odden, 1994).

(14) Kikongo (Ao, 1991)

a. /ku-toot-ila/ ↦[ku-toot-ila]
'to harvest for'
b. /ku-dumuk-ila/ ↦[ku-dumuk-ina
'to jump for'
c. /ku-dumuk-is-ila/↦[ku-dumuk-is-ina]
'to make jump for'

We can model this process schematically with the function from Fig. 3, in which any a following a b is output as a b, no matter how many c's intervene. We will use the string model in Figure 10, where b is the trigger, a is the target, and c intervenes (without blocking).

$\rtimes_1$	c_2	b_3	c_4	c_5	c_6	a_7	$\ltimes_8$

Figure 10: Model for the string *cbccca*

This time the positions labeled b in the output are identified by the formula in (15).

(15) $b'(x) \stackrel{\text{def}}{=} [\texttt{lfp}\ b(y) \vee A(p(y))](x) \wedge \neg c(x) \wedge \neg \ltimes(x)$

The set is built as follows:

$\varnothing$
$\{3\}$
$\{3, 4\}$
$\{3, 4, 5\}$
$\{3, 4, 5, 6\}$
$\{3, 4, 5, 6, 7\}$
$\{3, 4, 5, 6, 7, 8\}$

Note that all positions after the trigger are included in the set, but the formula for $b'(x)$ specifies that to be labeled b in the output a position has to both be in this set AND not be labeled c (or again $\ltimes$). So only the trigger and target are labeled b, giving the output string *cbcccb*.

6 Properties of QFLFP

6.1 Relation of QFLFP to other classes of functions

We now show some important properties of QFLFP. In particular, the structure of QFLFP concisely restricts its transductions to subsequential functions. Specifically, QFLFP transductions model the deterministic computation of the output string reading the input string left to right.

Remark 1 QFLFP $\subseteq$ MSO.

Proof: It is sufficient to show a translation into MSO for formulas of the form $[\texttt{lfp}\ \varphi(A, y)](x)$. We can replace any such formula for an equivalent MSO formula $(\exists X \forall y)[(\varphi(A/X, y) \rightarrow X(y)) \wedge X(x)]$, where $\varphi(A/X, y)$ is $\varphi(A, y)$ with each instance of $A(p^n(y))$ replaced with $X(p^n(y))$. □

The following shows that satisfaction of a QFLFP formula is closed under suffixation; that is, if a position in a string satisfies a formula, it will satisfy that formula regardless of how the string is suffixed. This shows that QFLFP is strictly less powerful than full MSO transductions.

Lemma 1 *For any* QFLFP *formula* $[\texttt{lfp}\ \varphi(A, y)](x)$, *any* $w \in \texttt{pref}(\rtimes\Sigma^*\ltimes)$, *and any* $1 \leq n \leq |w|$,

$$w \vDash [\texttt{lfp}\ \varphi(A, y)](x)[x \mapsto n]$$
implies
$$wv \vDash [\texttt{lfp}\ \varphi(A, y)](x)[x \mapsto n]$$

for all $v \in \texttt{suff}(\rtimes\Sigma^*\ltimes)$.

Proof: Let $\texttt{lfp}_w(F_\varphi)$ be the least fixed point with respect to F_φ for the domain D_w of (the model of) w; likewise $\texttt{lfp}_{wv}(F_\varphi)$ for wv. We show that $\texttt{lfp}_w = (\texttt{lfp}_{wv} \cap D_w)$; this means that $n \in \texttt{lfp}_w$ if and only if $n \in \texttt{lfp}_{wv}$ for all $1 \leq n \leq |w|$.

Consider the series $X^0_w = \varnothing$, $X^{i+1}_w = F_\varphi(X^i_w)$ from (7), relativized to the domain of w, and similarly $X^0_w = \varnothing$, $X^{i+1}_{wv} = F_\varphi(X^i_{wv})$ for wv. By definition $X^0_w = X^0_{wv} = \varnothing$.

It is then the case that $X^1_w = (X^1_{wv} \cap D_w)$. For any $1 \leq j \leq |w|$, $w \vDash \varphi(A, y)[A \mapsto \varnothing, y \mapsto j]$ implies $wv \vDash \varphi(A, y)[A \mapsto \varnothing, y \mapsto j]$ because they are equivalent with respect to the base cases: for $\varphi(A, y) = a(p^m(y))$, $w \vDash a(p^m(y))[y \mapsto j]$ iff $v \vDash a(p^m(y))[y \mapsto j]$ because $\texttt{pref}_j(w) = \texttt{pref}_j(wv)$; $w, v \nvDash A(p^m(y))[A \mapsto \varnothing, y \mapsto j]$, because this always evaluates to false. So $j \in X^1_w$ iff $j \in X^1_{wv}$ for $1 \leq j \leq |w|$.

We then consider X^i_w and X^i_{wv} for an arbitrary $i \geq 2$. By hypothesis, assume that $X^i_w = (X^i_{wv} \cap D_{wv})$. Then for any $1 \leq j \leq |w|$, because $\texttt{pref}_j(w) = \texttt{pref}_j(wv)$, $w \vDash \varphi(A, y)[A \mapsto X^i_w, y \mapsto j]$ iff $v \vDash \varphi(A, y)[A \mapsto X^i_{wv}, y \mapsto j]$. So $j \in X^{i+1}_w$ iff $j \in X^{i+1}_{wv}$, and thus $X^{i+1}_w = (X^{i+1}_{wv} \cap D_{wv})$.

Because we have shown this also for $i = 1$, it must then hold for any arbitrary i. Thus $\texttt{lfp}_w = (\texttt{lfp}_{wv} \cap D_w)$. □

Lemma 2 *For any* QFLFP *formula* $\varphi(x)$*, any* $w \in \texttt{pref}(\rtimes\Sigma^*\ltimes)$*, and any* $1 \leq n \leq |w|$*,*

$$w \vDash \varphi(x)[x \mapsto n] \text{ implies } wv \vDash \varphi(x)[x \mapsto n]$$

for all $v \in \texttt{suff}(\rtimes\Sigma^*\ltimes)$*.*

Proof: By recursion on the structure of $\varphi(x)$. If $\varphi(x) = a(p^m(x))$ for some $a \in \Sigma \cup \{\rtimes, \ltimes\}$, then $w(n) = wv(n) = a$. If $\varphi(x) = [\texttt{lfp}\ \psi(A, y)](x)$ and $w \vDash \varphi(x)[x \mapsto n]$ then $wv \vDash \varphi(x)[x \mapsto n]$ by Lemma 1. The cases for $\varphi(x) = \neg\psi(x)$ and $\varphi(x) = \psi_1(x) \vee \psi_2(x)$ then follow. □

The above means that for a QFLFP transduction T the output of a prefix $w \in \Sigma^*$ in the input remains constant, regardless of how we extend w. We formalize this output with $\texttt{out}_T(w) = w'_1 w'_2 ... w'_\ell$, where $\ell = |w|$ and each $w'_i = \gamma_1\gamma_2...\gamma_n$, such that $w \vDash \gamma_1^1(x)[x \mapsto i]$, $w \vDash \gamma_2^2(x)[x \mapsto i]$, ..., $w \vDash \gamma_n^n(x)[x \mapsto i]$ for $\gamma_1^1(x), \gamma_2^2(x), ..., \gamma_n^n(x) \in T$.

The following is thus important in establishing the tails of w.

Corollary 1 *For any string* $w \in \Sigma^*$*, a* QFLFP*-definable transduction* T *and its function* $f = f(T)$*,* $\texttt{out}(w)$ *is a prefix of* $\texttt{lcp}(f(w\Sigma^*))$*.*

Proof: Let $\texttt{out}_T(w) = w'_1 w'_2 ... w'_\ell$ as above. The corollary follows from Lemma 2, because at each position i in w, i will satisfy the same formulas no matter how w is extended, so w'_i will be output no matter how w is extended. Thus, at the least, $w'_1 w'_2 ... w'_\ell$ will be output. □

The following establishes the set of 'reach strings' for $\varphi(x) \in T$; that is, strings w such that some $w\sigma$ will satisfy $\varphi(x)$. This simulates the set of strings that will reach a particular state in a FST.

Definition 5 *For* $\varphi(x) \in T$*, let*

$$L_\varphi \stackrel{def}{=} \left\{ w \;\middle|\; w\sigma \vDash \varphi(x)[x \mapsto |w\sigma|] \text{ for some } \sigma \in \Sigma \right\}$$

The following follows from the fact that QFLFP $\subseteq$ MSO.

Remark 2 L_φ *is regular.*

For $\varphi(x) \in T$, let P_φ be the finite partition on Σ^* induced by the equivalence relation of L_φ. Then for $T = \{\varphi_1(x), ..., \varphi_n(x)\}$, let

$$P_T \stackrel{\text{def}}{=} P_{\varphi_1} \otimes P_{\varphi_2} \otimes ... \otimes P_{\varphi_n}$$

This is the partition of Σ^* into sets of strings belonging to the same equivalence class for every L_{φ_i}. As each L_{φ_i} is regular, it follows that P_T is finite.

Lemma 3 *For some* QFLFP $T = \{\varphi_1(x), ..., \varphi_n(x)\}$ *and* $f = f(T)$*, for any* $w, v \in \Sigma^*$ *and* $A \in P_T$*,*

$$w, v \in A \;\leftrightarrow\; w \equiv_f v$$

Proof: ($\rightarrow$) Recall (from §2) that for any $\varphi(x) \in T$ and $B \in P_\varphi$, $w, v \in B$ iff for all $z \in \Sigma^*$, $wz \in L_\varphi$ iff $vz \in L_\varphi$. Because P_T is the pairwise intersection of all such sets, for any $A \in P_T$, $w, v \in A$ if and only if for all $z \in \Sigma^*$ and for all $\varphi(x) \in T$, $wz \in L_\varphi \leftrightarrow vz \in L_\varphi$.

Thus, for $w, v \in A$ in P_T if and only if for all $z \in \Sigma^*$ and for all $\varphi(x) \in T$,

$$\begin{aligned} wz\sigma \vDash \varphi(x)[x \mapsto |wz\sigma|] \quad &\leftrightarrow \\ vz\sigma \vDash \varphi(x)[x \mapsto |vz\sigma|] \end{aligned}$$

Consider then any $w, v \in A$ for some $A \in P_T$ and any $u = \sigma_1\sigma_2...\sigma_m$. It follows that for each σ_i $w\sigma_1...\sigma_i \vDash \varphi(x)[x \mapsto (|w| + i)]$ if and only if $v\sigma_1...\sigma_i \vDash \varphi(x)[x \mapsto (|v| + i)]$.

Then consider $u' = u'_1 u'_2 ... u'_n$ where each $u'_i = \gamma_1^1\gamma_2^2...\gamma_k^k$ s.t. $w\sigma_1...\sigma_i \vDash \gamma_j^j(x)[x \mapsto (|w| + i)]$ for $\gamma_1^1(x), ..., \gamma_k^k(x) \in T$. From Lemma 2 we know

$$f(wu) = \texttt{out}_T(w)u' \text{ and } f(vu) = \texttt{out}_T(v)u'.$$

From Corollary 1 it is also the case that $\texttt{out}_\texttt{T}(w)$ is a prefix of $\texttt{lcp}(f(w\Sigma^*))$ and $\texttt{out}_\texttt{T}(v)$ is a prefix of $\texttt{lcp}(f(v\Sigma^*))$. Let u'_p be the portion of u' that is in $\texttt{lcp}(f(w\Sigma^*))$; that is, $u' = u'_p u'_s$ where $\texttt{lcp}(f(w\Sigma^*)) = \texttt{out}_\texttt{T}(w)u'_p$. Because the above establishes that w and v share the same output suffixes, it is also the case that $\texttt{lcp}(f(v\Sigma^*)) = \texttt{out}_\texttt{T}(v)u'_p$. Thus any (u, u'_s) is in $\texttt{tails}_f(w)$ if and only if it is also in $\texttt{tails}_f(v)$.

Thus $w, v \in A$ implies that $\texttt{tails}_f(w) = \texttt{tails}_f(v)$.

($\leftarrow$) The reverse is clear by considering that $(u, u'_1 u'_2 ... u'_n) \in \texttt{tails}_f(w)$ if and only if for each $u_i = \gamma_1^1\gamma_2^2...\gamma_k^k$, $w\sigma_1...\sigma_i \vDash \gamma_j^j(x)[x \mapsto (|w| + i)]$ for each $\gamma_j^j(x)$. If this is also the case for v then it must be the case that $w, v \in A$ for the same $A \in P_T$. □

Theorem 2 QFLFP $\subseteq$ SUBSEQ.

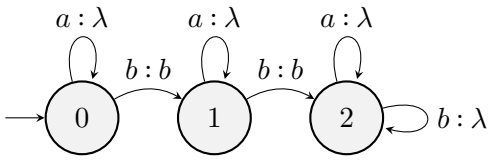

Figure 11: An OSL_3 function we conjecture not to be QFLFP.

Proof: From Lemma 3 and the fact that P_T is finite. □

Lemma 4 QFLFP $\not\subseteq$ OSL

Proof: This is witnessed by the function of the SFST in Fig. 3 (which turns any a following a b into an a, regardless of any intervening c's), which was shown not to be OSL, but was shown in Sec. 5 to be QFLFP-definable. □

Conjecture 1 OSL, SUBSEQ $\not\subseteq$ QFLFP

Consider the OSL function in Figure 11, which deletes any a and all except for the first two b's.[3] This is OSL_3, because whether or not we delete a b depends on whether or not we have output 2 b's previously. Now consider a formula $b'(x)$ that is true for exactly the first two b's. For the first, we can write $b(x) \wedge \neg[\texttt{lfp}(b(y) \vee A(p(y)))](x)$, the latter `lfp` disjunct identifying any elements following a b. To identify exactly the second b, we would have to include reference to the first b in another `lfp` predicate, thus embedding one `lfp` statement in another. We conjecture that this embedding of one `lfp` formula in another is necessary; that is, there is no QF formula $\varphi(A, y)$ such that $[\texttt{lfp}\ \varphi(A, y)](x)$ can identify the second b.

6.2 Equivalence of QF and ISL

Finally, we show that the QF fragment of QFLFP describes exactly the ISL functions. We first show that QF formula can only distinguish positions in a string based on the previous $k-1$ symbols for some k.

Lemma 5 *Let T be a* QF *transduction, and let k be the value for which $p^{k-1}(x)$ appears in some $\gamma^c(x)$ in T and for any other $p^j(x)$ that appears in some $\gamma^c(x)$ in T, $j < k$. Consider two strings $w, v \in \texttt{pref}(\rtimes\Sigma^*\ltimes)$ such that $\texttt{suff}_{k-1}(w) = \texttt{suff}_{k-1}(v)$; let ℓ_w denote $|w|$ and likewise ℓ_v for $|v|$. For any $\gamma^c(x)$ in T,*

$$w \vDash \gamma^c(x)[x \mapsto \ell_w]$$
if and only if
$$v \vDash \gamma^c(x)[x \mapsto \ell_v]$$

Proof: Let $u = \texttt{suff}_{k-1}(w) = \texttt{suff}_{k-1}(v)$ and ℓ_u denote $|u|$. If $\gamma^c(x) = a(p^j(x))$ for some $a \in \Sigma \cup \{\rtimes, \ltimes\}$, then $w \vDash \gamma^c(x)[x \mapsto \ell_w]$ iff $w(\ell_w - j) = a$. Since $j < k$, this implies that $u(\ell_u - j) = a$ and thus also that $v \vDash \gamma^c(x)[x \mapsto \ell_v]$. Clearly the reverse implication holds as well. Since w and v are equivalent with respect to satisfaction of the atomic formulae for QF, this then extends to the general case for $\varphi(x)$ based on induction on the structure of QF.□

Theorem 3 QF = ISL

Proof: ($\rightarrow$) From Lemma 3 we know that w and v have the same set of tails; thus Lemma 5 shows that $\texttt{suff}_{k-1}(w) = \texttt{suff}_{k-1}(v)$ implies $w \equiv_f v$. Thus QF $\subseteq$ ISL.

($\leftarrow$) For ISL $\subseteq$ QF, we construct a QF transduction for the canonical transducer for an ISL function. As stated in §3, each state in an ISL_k SFST represents a $k-1$ suffix. So we can construct an equivalent QF transduction T as follows. For a string $w = \sigma_1\sigma_2...\sigma_k$ we can write $\varphi_w(x) = \sigma_1(p^{k-1}(x)) \wedge \sigma_2(p^{k-2}(x)) \wedge ... \wedge \sigma_{k-1}(p(x)) \wedge \sigma_k(x)$. For each symbol $\gamma \in \Gamma$, we set $\gamma^i(x) = \varphi_{w_1}(x) \vee \varphi_{w_2}(x) \vee ... \vee \varphi_{w_n}(x)$, where $w_1, w_2, ..., w_n$ is the exhaustive set of strings $w_j = q_j\sigma_j$ for which a transition $\delta(q_j, \sigma_j) = (v_j, r_j)$, exists where $\gamma = v_j(i)$. So for any position in the input string that exercises a transition $\delta(q, \sigma) = (v, r)$ in the ISL_k machine, it will also satisfy a sequence $\gamma_1^1(x), \gamma_2^2(x), ..., \gamma_m^m(x)$ such that $\gamma_1\gamma_2...\gamma_m = v$. □

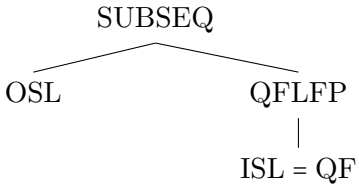

Figure 12: Hierarchy of the relevant function classes

Fig. 12 summarizes the results discussed so far.

[3] We thank Shiori Ikawa for this example.

6.3 Left- and right-subsequential functions

We have so far abstracted away from the fact that SUBSEQ can instead be characterized as the *left*-subsequential functions, with the *right*-subsequential functions being the reversal of some left-subsequential function (Mohri, 1997). Formally, a function f is right-subsequential iff $f = \{(w^r, v^r) | f'(w) = v\}$, where w^r is the reverse of string w, for some left-subsequential function f'. Equivalently, the right-subsequential functions are those that can be described by a SFST reading and writing strings right-to-left.

In terms of QFLFP, when the input signature $\mathcal{I}$ contains the predecessor function p we obtain left-subsequential functions. As is perhaps clear from the above discussion, if we instead use an identical signature with a successor function s, we obtain the right-subsequential functions.

Crucially, inclusion of QFLFP in one of the subsequential classes relies on using *either* s *or* p, but not *both*. Consider the following transduction defined with a signature including both s and p.[4]

(16) a. $a'(x) \stackrel{\text{def}}{=} a(x) \wedge \big(\rtimes(p(x)) \rightarrow [\texttt{lfp}\, b(y) \vee A(s(y))](x) \big)$

b. $b'(x) \stackrel{\text{def}}{=} b(x) \wedge [\texttt{lfp}\, b(y) \vee A(p(y))](x)$

c. $c'(x) \stackrel{\text{def}}{=} \big(a(x) \wedge \rtimes(p(x)) \wedge \neg[\texttt{lfp}\, b(y) \vee A(s(y))](x)\big) \vee \big(b(x) \wedge \neg[\texttt{lfp}\, b(y) \vee A(p(y))](x)\big)$

This transduction takes strings of a's and b's and outputs as a c 1) the first a if and only if there are no b's in a string; or 2) the first b in the string. The definition of $c'(x)$ in (16-c) outputs a c for an input initial a that is not followed by a b (the first disjunct; $a(x) \wedge \rtimes(p(x)) \wedge \neg[\texttt{lfp}\, b(y) \vee A(s(y))](x)$) or an input b that is not preceded by another b (the second disjunct; $b(x) \wedge \neg[\texttt{lfp}\, b(y) \vee A(p(y))](x)$). The definitions in (16-a) and (16-b) output a's and b's, respectively, for any a and b that does not meet the conditions in (16-c) for outputting a c. A (nondeterministic) FST for this function is given in Fig. 13; examples of the mapping are given below in (17).

[4]We thank Nate Koser for this example.

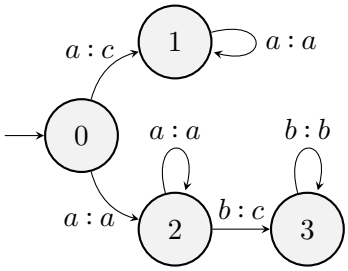

Figure 13: A properly regular function definable using QFLFP with both successor s and predecessor p functions.

(17) $aaaaaa \mapsto caaaaa$
$aababa \mapsto aacaba$

It can be shown that this function is not subsequential, and thus there is no equivalent SFST for the FST in Fig 13. The reason is made clear by the definition in (16): the transduction looks *ahead* to the right to check if an initial a should be output as a c, but looks *behind* to the left to check if a b is the first b. These cannot both be accomplished by a deterministic FST that reads either right-to-left or left-to-right. Thus, restricting the signature to either p or s is crucial to capturing the behavior of subsequential FSTs.

7 Discussion

We have introduced a class of functions called QFLFP that are defined as graph interpretations using quantifier-free first order logic formulas augmented with a least fixed point operator. In this section we discuss some of the implications of using this class to model phonological processes and highlight a few important areas of future work.

One of the main advantages of such logical characterizations of phonological processes is that they enable a unified approach to both linear and non-linear representations. The graphs used to represent strings in this approach can be extended to represent additional structure used in phonological theory such as autosegmental representations and feature geometry (Goldsmith, 1976; Clements, 1985), syllable constituents (Selkirk, 1984), or metrical structure (Hayes, 1995). This flexibility allows us to directly apply the notion of subsequentiality to other types of representations, which in turn enables more direct comparison among types of phonological phenomena and theories of phonological representation, as we can change the models to accommodate the added

structure but maintain the restrictions on the logical formalism (see Chandlee and Jardine, 2019, for an example). This was not as straightforward with finite-state automata.

We have argued that the QFLFP class provides an attractive fit to the observed typology of phonological functions, capturing both local and long-distance phenomena, which the previously defined ISL and OSL classes cannot do without introducing the mechanism of string markup. Compared to the subsequential functions, QFLFP can describe functions dependent on even/odd parity in a local way—such as assigning iterative stress—but not in an unbounded way (e.g., identify every other vowel regardless of how many consonants intervene). The subsequential functions can describe such unbounded even/odd phenomena, which are hypothesized to not exist in phonology. In this way QFLFP appears to be a better fit to the typology compared to the already restrictive subsequential functions.

Future work will aim to identify the limits of QFLFP and the kinds of phonological processes and process interactions that it cannot describe. Such cases will point to the ways the class of functions can be extended and/or modified to provide an even better fit to the range of attested phenomena without including logically possible but unattested patterns. For example, what is the expressivity of QFLFP when two-place predicates (see Koser et al., 2019) or embedded `lfp` operators are allowed?

There are also several theoretical questions left to be addressed. Are QFLFP functions closed under composition? We conjecture that they are not, as per the discussion of the OSL function in Fig. 11 that we conjecture not to be QFLFP. The reason is that functions in which `lfp` operators are embedded in other `lfp` operators appear to be strictly more expressive than those which are not. We also leave an abstract characterization of the QFLFP functions to future work. Such a characterization would lead to a definitive answer to the conjecture that SUBSEQ is a proper superset of QFLFP, as well as whether or not QFLFP is closed under composition. Finally, this paper has considered QFLFP defined over string signatures with only a single ordering function p. The same logic defined over string signatures defined with both p and the successor function s will almost certainly be more expressive, but how much more expressive is an interesting question for future work.

8 Conclusion

QFLFP graph interpretations combine the restrictiveness of quantifier-free first order logic with an operator that can recursively reference the output structure. This allows us to model phenomena beyond the reach of the ISL functions, including iterative spreading processes and long-distance agreement. This class of functions appears to cross-cut several subregular function classes that have previously been applied to the modeling of phonological processes. Because it is still a subset of the subsequential functions, however, it is learnable from positive data. This combination of desirable properties incidates that QFLFP is an important step toward the goal of identifying and understanding the computational nature of phonological processes.

References

Benjamin Ao. 1991. Kikongo nasal harmony and context-sensitive underspecification. *Linguistic Inquiry*, 22(2):193–196.

Jane Chandlee. 2014. *Strictly Local Phonological Processes*. Ph.D. thesis, University of Delaware.

Jane Chandlee, Rémi Eyraud, and Jeffrey Heinz. 2015. Output strictly local functions. In *Proceedings of the 14th Meeting on the Mathematics of Language (MoL 2015)*, pages 112–125, Chicago, USA. Association for Computational Linguistics.

Jane Chandlee and Jeffrey Heinz. 2012. Bounded copying is subsequential: implications for metathesis and reduplication. In *Proceedings of SIGMORPHON 12*.

Jane Chandlee and Jeffrey Heinz. 2018. Strict locality and phonological maps. *LI*, 49:23–60.

Jane Chandlee and Adam Jardine. 2019. Autosegmental strictly local functions. *Transactions of the Association for Computational Linguistics*, 7:157–168.

Jane Chandlee and Steven Lindell. in prep. A logical characterization of strictly local functions. In Jeffrey Heinz, editor, *Doing Computational Phonology*. OUP.

G. N. Clements. 1985. The geometry of phonological features. *Phonology Yearbook*, 2:225–252.

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

Emmanual Filiot. 2015. Logic-automata connections for transformations. In *Logic and Its Applications (ICLA)*, pages 30–57. Springer.

Emmanuel Filiot and Pierre-Alain Reynier. 2016. Transducers, logic, and algebra for functions of finite words. *ACM SIGLOG News*, 3(3):4–19.

John Goldsmith. 1976. *Autosegmental Phonology*. Ph.D. thesis, Massachussets Institute of Technology.

Bruce Hayes. 1995. *Metrical stress theory*. Chicago: The University of Chicago Press.

Jeffrey Heinz and Regine Lai. 2013. Vowel harmony and subsequentiality. In *Proceedings of the 13th Meeting on the Mathematics of Language (MoL 13)*, pages 52–63.

Mans Hulden. 2009. *Finite-State Machine Construction Methods and Algorithms for Phonology and Morphology*. Ph.D. thesis, University of Arizona.

Adam Jardine. 2016. *Locality and non-linear representations in tonal phonology*. Ph.D. thesis, University of Delaware.

R.M. Kaplan and M. Kay. 1994. Regular models of phonological rule systems. *Computational Linguistics*, (20):371–387.

Nathan Koser, Christopher Oakden, and Adam Jardine. 2019. Tone association and output locality in nonlinear structures. In *Supplemental proceedings of the 2018 Annual Meeting on Phonology*. Linguistics Society of America.

Leonid Libkin. 2004. *Elements of Finite Model Theory*. Berlin: Springer-Verlag.

Huan Luo. 2017. Long-distance consonant agreement and subsequentiality. *Glossa: A Journal of General Linguistics*, 2(1):52.

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

David Odden. 1994. Adjacency parameters in phonology. *Language*, 70(2):289–330.

J. Oncina, J. García, and E. Vidal. 1993. Learning subsequential transducers for pattern recognition interpretation tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (15:5):448–457.

Farid M. Onn. 1980. *Aspects of Malay Phonology and Morphology: A Generative Approach*. Kuala Lumpur: Universiti Kebangsaan Malaysia.

Amanda Payne. 2017. All dissimilation is computationally subsequential. *Language: Phonological Analysis*, 93(4):e353–e371.

Marcel Paul Schützenberger. 1977. Sur une variante des fonctions séquentielles. *Theoretical Computer Science*, 4:47–57.

Elisabeth Selkirk. 1984. On the major class features and syllable theory. In Morris Halle, Mark Aronoff, and R.T. Oehrle, editors, *Language sound structure: Studies in phonology*. Cambridge, Mass.: MIT Press.