This printout has been approved by me, the author. Any mistakes in this printout will not be fixed by the publisher. Here is my signature and the date: _____

Learning Underlying Representations and Input-Strictly-Local Functions

Wenyue Hua, Adam Jardine, Huteng Dai
Rutgers University

1. Introduction

The simultaneous inference of underlying representations (URs) and a phonological grammar from alternating surface representations (SRs) in a morphological paradigm is a core problem in phonological learning that only recently has seen progress (Tesar, 2014; Cotterell et al., 2015; Rasin et al., 2018). This paper proposes a learning algorithm that infers URs and phonological processes from SRs based on the hypothesis that phonological generalizations belong to restrictive *subregular* regions in the Chomsky Hierarchy (Heinz, 2018). We give a procedure that, given sequences of morphemes paired with SRs, learns URs and a phonological grammar that is an *input strictly local* (ISL; Chandlee, 2014; Chandlee & Heinz, 2018) function. ISL functions are exactly those which make changes in the output with respect to the local information in the input. For now, the procedure is restricted to *simplex* ISL processes; that is, those exhibiting a single change. However, this illustrates that restrictive computational principles, combined with major principles in phonological analysis, allow for significant progress in understanding how phonological grammars and URs are learned.

The paper is organized as follows. Section 2 briefly introduces the paradigm of the learning algorithm. Section 3 discusses the computational structure encoded in the learner. Section 4 is a detailed explanation of the algorithm with a simple example as illustration. Section 5 compares this algorithm with other algorithms and presents its advantages. The last section concludes the paper.

2. Overview

A phonological process can be analyzed as the application of two functions: a function ℓ that maps each morpheme to its UR, and a function ϕ that maps each UR to its SR. The observed SRs are the outputs of the composition of both ℓ and ϕ. Given the concatenation of morphemes $m_1m_2m_3...m_n$, its SR is $\phi(\ell(m_1m_2m_3...m_n))$. With the inputs and outputs given, the ultimate task is to learn the two intertwined functions. The hypotheses of URs and phonological processes are co-dependent: a difference in the inferred URs will incur a difference in the proposed phonological processes and vice versa. Therefore the algorithm needs to infer URs and phonology jointly. Based on this intuition, we propose a new paradigm of learning: exact inference based on two interacting *finite-state transducers* (FSTs). The first transducer encodes ℓ, which takes morphemes as inputs and outputs URs; the second transducer encodes ϕ, which takes URs as inputs and outputs SRs.

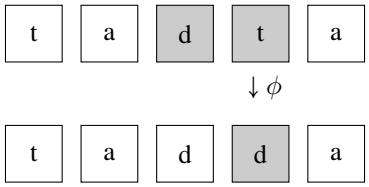
The algorithm takes as input pairs of the concatenations of morphemes and their SRs. The algorithm is assumed to have a morphological analysis of given SRs, due to which we can deviate from the complexity involved in lexical semantic learning, morphosyntactic learning and the morphological segmentation (Tesar, 2014). The primary goal is to use computational properties of the target phonology to understand how the learning of these patterns (by either machines or humans) is possible, and how the learning proceeds. Therefore an FST representing a specific class of subregular function is given to the learner. If the target function is a regular function belonging to a class of sub-regular functions that shares a particular structure, there are known techniques for learning it from positive data (Jardine et al., 2014). The ISL class is one such class (Jardine et al., 2014) to which a great many phonological processes belong (Chandlee & Heinz, 2018; Chandlee et al., 2018).

Cascadilla Proceedings Project
Completed October 10, 2020

Our goal is a provably correct algorithm whose behavior in the general case is well-understood (Heinz, 2010; Tesar, 2014), rather than one whose behavior can only be tested by running simulations (Hayes & Wilson, 2008; Rasin et al., 2018). The examples we use to demonstrate the algorithm are thus simplified in order to focus on the general principles that underlie the learner.

3. Input-Strictly-Local functions and learnability

An ISL function computes the output of any input element in a string by looking at a bounded number of input elements coming before or after it. In other words, given any input segment, segments in a fixed length window around it can completely determine its output, which is similar to the idea of n-gram. For example, the diagram below shows a progressive voicing assimilation process as a function ϕ applying to an input /dt/ sequence.



Note that to compute the output of the highlighted /t/ as [d], only the preceding /d/ is relevant—the preceding /a/, for example, is not. As ϕ thus depends only on sequences of length 2 (specifically, a /dt/ sequence), it is ISL_2.

For any ISL function, it is ISL_k if each input segment's output depends on at most k inputs (including the segment itself) around it. Although the ISL class is a computationally restrictive class of functions, it has been shown to be empirically relevant for phonology. Chandlee (2014) analyzed approximately 5500 patterns from about 500 hundred languages in P-Base database (Mielke, 2004). She found 94% of the patterns in the database are ISL. The remaining 6% includes some suprasegmental processes and iterative and long-distance processes such as vowel and consonant harmony.

The ISL class is also learnable from positive data consisting of example input-output pairs (Chandlee, 2014; Chandlee et al., 2014; Jardine et al., 2014). This result is crucial because the learnability of a composition of two ISL functions is possible only if one ISL function is proven learnable. In particular, Jardine et al. (2014) showed that by providing an abstract structure of the target function to the learner as *a priori* knowledge, the algorithm Structured Onward Subsequential Function Inference Algorithm (SOSFIA) can learn any function characterizable by deterministic FSTs. With the structure of the FST given, the algorithm only needs to learn the output of each transition.

The current project adopts this learning strategy in which the structure of the target functions are given to the learner *a priori*. However, the current project is crucially different from previous work in that it learns two functions, a phonology function ϕ and a lexicon function ℓ, from the input-output of their composition $\phi \circ \ell$. The inputs of ℓ are given while the inputs of ϕ are part of what needs to be learned; the output of ϕ are given while the outputs of ℓ are part of what needs to be learned. However, since Jardine et al. (2014)'s work can learn any class of functions that can be represented by varying the outputs on some deterministic FST, the current project can in principle be extended to learn phonology in non-ISL classes of functions.

4. General Principles and Learning Algorithm

The general procedure of the algorithm is largely motivated by phonological analysis in generative linguistics. It strictly follows the following principles of generative phonological analysis:

(1) (a) Each morpheme has a single UR.
(b) Among allomorphs of a morpheme, the one that occurs in most environments is the UR.
(c) Different allomorphs are in complementary distribution.

While there are of course exceptions to these basic principles, they are a reasonable place to start. Based on these fundamental principles, we give an algorithm that can discover the URs and phonological

transformations jointly. The algorithm encodes the lexicon function ℓ and the phonology function ϕ by two FSTs. It starts out proposing two initial hypothesized functions, then detecting the inconsistencies in lexicon hypotheses based on *Principle (1a)* to find what modifications are necessary, selecting URs among allomorphs based on *Principle (1b)* and learning the phonological transformation based on *Principle (1c)*, in the end updating the transducers. The rest of the section explicates how these three principles are incorporated in the design of the algorithm with learning a progressive assimilation process as a toy example.

Table 1 is a toy example data set. The *alphabet* (or inventory of segments) Σ is $\{t, a, d\}$. The left table gives the morphological composition and SRs of words, which is the input to the algorithm. The right is the UR of each morpheme and the phonological process that generates the SRs based on UR concatenations, both of which are the target outputs of the algorithm.

Morph.	**SR**	**Morph.**	**SR**
$r_1 s_1$	[tatta]	$r_3 s_1$	[ata]
$r_1 s_2$	[tadda]	$r_3 s_2$	[ada]
$r_1 s_3$	[tata]	$r_3 s_3$	[aa]
$r_2 s_1$	[tadda]	$r_4 s_1$	[taddta]
$r_2 s_2$	[tadda]	$r_4 s_2$	[taddda]
$r_2 s_3$	[tada]	$r_4 s_3$	[tadda]

Morph.	**UR**	**Morph.**	**UR**
r_1	/tat/	s_1	/ta/
r_2	/tad/	s_2	/da/
r_3	/a/	s_3	/a/
r_4	/tadt/		

phonological transformation: $t \rightarrow d \,/ d$ ___

Table 1: progressive assimilation: example inputs and outputs

4.1. Step-wise Procedure of the Algorithm

Algorithm 1 provides the algorithm in pseudocode and the rest of the section illustrates how the algorithm works in detail with the example above.

The **first** step of the algorithm, in line 1 of Alg. 1, is to establish two initial hypotheses for ℓ and ϕ using FSTs. These FSTs implement the notion that the initial hypotheses of the learner assume that the URs are identical to the surface forms (but are dependent on morphological context) and that the phonology function is entirely faithful.

The hypothesis for the lexicon function is realized by building a *prefix tree transducer (*`PTT`*)* T_ℓ and the hypothesis for phonology is realized by an ISL transducer T_ϕ. The `PTT` T_ℓ is a finite-state representation of the input data with a tree whose branches represent shared *prefixes* (initial sequences) in the input. It represents a method of parsing the data. For example, the branch from state 0 to state 1 represents several hypothesized morpheme-UR pairs[1]: the morpheme input r_1 and the hypothesized UR tat (the input r_3 and the hypothesized UR a and the input r_4 and the hypothesized UR $tadd$). The parsing is realized by computing *longest common prefix* (`lcp`; the longest initial shared sequence) of the surface forms that share an initial morpheme. For example, $r_1 s_1$ ($tatta$), $r_1 s_2$ ($tatda$) and $r_1 s_3$ ($tata$) all begin with the morpheme r_1. The longest initial sequence shared by the three SRs is *tat*. Thereby the SR of r_1 is computed to be tat, which also becomes the algorithm's initial guess for the UR of r_1.

Notice, however, that different hypotheses for URs appear after different preceding morphemes. For example, the hypothesized UR for s_1 is *ta* when following r_1, whereas it is *da* when following r_2. In this way, the `PTT` keeps track of the morphological context of each SR. The goal of the algorithm is to replace the morphological context with the *phonological* context of the SRs.

The phonology transducer T_ϕ represents the phonological contexts. In an ISL$_k$ transducer, the states represent the previous $k-1$ segments. They can also be understood as the phonological environment for each segment. In an ISL$_k$ phonology transformation, the output of each underlying segment is determined by k many segments in total, including the underlying segment itself. Therefore the $k-1$ segments consist of the phonological environment for a phonology transformation if the output string differ from the underlying segment.

[1] For clarity only the branches for r_1 and r_2 are shown in the `PTT` on the left. (The branches for r_3 and r_4 would be similar to that for r_1.)

1 Build PTT T_ℓ based on data, initialize T_ϕ to the identity function
2 Find some m such that there are two distinct outputs w_1, w_2 in T_ℓ for m
3 Initialize E_1, E_2 to $\{\}$
4 **for** $m_1^1 ... m_x^1$ *preceding* m *when* w_1 *is output* **do**
5 | Let w_1' bet the output of $m_1^1 m_2^1 ... m_x^1$
6 | Add the state in T_ϕ that w_1' reaches to E_1
7 **for** $m_1^2 ... m_y^2$ *preceding* m *when* w_2 *is output* **do**
8 | Let w_2' bet the output of $m_1^2 m_2^2 ... m_x^2$
9 | Add the state in T_ϕ that w_2' reaches to E_2
10 Let A be the smaller of E_1, E_2 and let $w_A \in \{w_1, w_2\}$ be its associated output
11 Let B be the larger of E_1, E_2 and let $w_B \in \{w_1, w_2\}$ be its associated output
12 **if** $|A| > 2$ *or* $|A| = |B|$ **then**
13 | restart the algorithm from right
14 Let e be the single member of $A - (A \cap B)$ or A
15 Let τ be the first segment of w_B, and u the rest of w_B
16 Let ρ be w_A with u removed from the end
17 Set the output of τ to ρ following state e in T_ϕ
18 **for** m *with multiple outputs in* T_ℓ **do**
19 | Let w_B be the output of m after $b \in B - e$
20 | **if** *Let* α *be the single member of* $A \cap B$ **then**
21 | | **if** $\alpha = e$ **then**
22 | | | undo ϕ for $\{m' \mid T_\ell(m')^{-1} T_\ell(m'm)^a = w_B, T_\ell(m')$ ends in $\alpha\}$
23 | | **else if** $\alpha \neq e$ **then**
24 | | | undo ϕ for $\{m' \mid T_\ell(m')^{-1} T_\ell(m'm) \neq w_B, T_\ell(m')$ ends in $\alpha\}$
25 | Set the output of m to be w_B

Algorithm 1: Pseudocode for the learner

[a] The operation $^{-1}$ is string subtraction.

For example, T_ϕ in Fig. 2 is an ISL_2 transducer whose states represent one $(k - 1 = 1)$ single segment. The input to the FST is underlying segments and the output is strings of surface segments. The output string is computed by reading input segments one by one and returning their corresponding outputs in the transducer. Thus, all ISL_2 functions (over the same inventory of underlying segments) have the same state structure. That is, they all encode environments that consist of the immediately preceding segment. The initial hypothesis is that the phonology is entirely faithful; in technical terms, T_ϕ first represents the identity function. The algorithm can then modify this hypothesis to other ISL_2 functions simply by changing the outputs of the transitions.

The **second** step of the algorithm, in line 2 of Alg. 1, is to detect inconsistency in the initial hypothesis of ℓ. Based on *Principle (1a)*, each morpheme should have only one UR and thereby inconsistencies in the initial hypothesis indicates the SR is not identical to the UR, *i.e.* that some phonological process is required to generate different observed representations. This can be observed in T_ℓ in Fig. 1 where there are multiple different transitions outputting different strings when given the same input morpheme m. In the example, there is one morpheme with inconsistent UR hypotheses: s_1. Its hypothesized UR is ta after state 1 while it is da after state 2. In other words, it surfaces as ta after tat, a and $tadd$ but surfaces as da after tad.

The **third** step of the algorithm, from lines 4 through 9 of Alg. 1, is to collect the environments for the allomorphs. In T_ℓ a unique string needs to be proposed as the output for the morpheme transition, which is the true UR. To find the true UR among the allomorphs, according to *Principle (1b)*, the one that occurs in more diverse environment is the UR. Therefore the algorithm needs to collect the environment for the allomorphs in order to learn URs. Since the algorithm knows *a priori* that the target phonology is ISL_2, the environment for each allomorph is just a subset of states in the transducer. For the allomorph ta, it comes after strings tat, a and tad and for da, it comes after tad. To find the environment sets systematically, the algorithm finds the states in T_ϕ that are reached by the surface forms of the morphemes immediately preceding s_1. In the example, the environment set for ta is $\{t, a, d\}$ and that for da it is $\{d\}$.

The **fourth** step of the algorithm, in lines 10 through 13, is to determine which allomorph is the UR based on the collected environment sets E_1 and E_2 for the allomorphs w_1 and w_2, respectively. Based on *Principle (1b)*, the allomorph that occurs in most diverse environment is the UR, so the algorithm sets A to be the smaller of E_1 and E_2 and B to be the larger. Naively, the allomorph w_B associated with B will then be the UR. However, in two situations the environment sets collected by the algorithm will not be the target phonological environment. The first is when the phonological transformation is regressive,

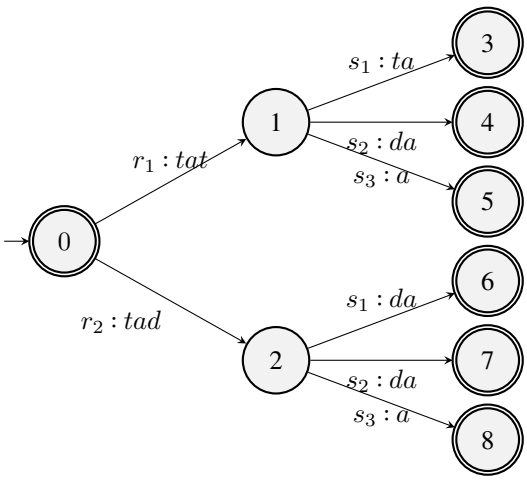


Figure 1: (Partial) T_ℓ as initial hypothesis for ℓ

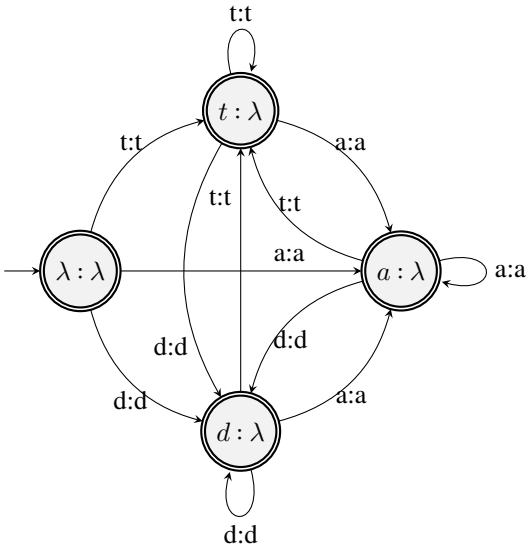


Figure 2: ISL$_2$ T_ϕ as initial hypothesis for ϕ

and the second is when the process involves opacity (in the sense of non-surface true phonology). When the process is regressive, the environment sets collected with the learner reading the input strings from left to right is obviously not the phonological environment. When the process involves opacity, the environment sets may contain segments that are not surface true.

Only when the process is regressive the two environment sets could be of a similar cardinality. In simple progressive processes, the size of the environment set for the non-UR allomorph is at most 2: one for the triggering segment, and one for non-surface true contexts (more on the latter below). However, if the process is regressive, its size would be large, as in a regressive process the preceding context is irrelevant. Detailed explanation and analysis are omitted for space reasons, but the algorithm can use this fact to discover that the process is regressive, and restart and read the input strings from right to left.[2] In other words, regressive processes can be learned using the same procedure as progressive processes.

When the process involves opacity, since one single process can involve only one underlying segment and surface segment alternation, there can be (at most one) additional segment in the environment sets. This is discussed in more detail below.

In this example, the smaller environment set is $\{d\}$ and the larger one is $\{t, a, d\}$. Since $|\{d\}| < 2$, the algorithm chooses da as w_A, i.e. the surface allomorph. As ta is associated with a more diverse environment, the algorithm chooses it as w_B, the proposed UR.

After learning the UR, the **fifth** step of the algorithm, in lines 14 through 16, is to learn the phonological process which generates observed allomorphs. In order to propose a phonological process, the algorithm has to learn the environment, the target, and the structural change of the process. Recall that in the current paradigm, the environments are represented by states in T_ϕ. As here we are assuming T_ϕ is ISL$_2$, the possible environments are the preceding segment.

In order to learn the triggering segment, the algorithm looks for the environments preceding allomorphs that are distinct from the UR. According to *Principle (1c)*, the environments where SR surfaces faithfully and the environment where SR differs from UR should be in complimentary distribution. In terms of the algorithm, $A \cap B = \varnothing$. Then the environment for the process is simply the segment preceding the non-UR allomorph. In this case, then $A = \{e\}$, where e is the environment for the process. This is true for the example: $A = \{d\}$, where d is the segment that triggers the change.

However, $A \cap B$ is not empty: d is also in B. This is because ϕ in this case is not surface true, as is often the case for phonological processes (McCarthy, 1999). Opacity may interfere so that the observed strings of segments are not the actual phonological environment of a transformation. In other words, the

[2] Deterministic FSTs that read the input right-to-left are known as *right-subsequential* transducers, and are a standard variant of FSTs (Mohri, 1997).

observed environment sets may not be the true underlying environment sets.

There are two cases where the observed environment sets differ from the true ones, as there are two types of opacity that can occur with a single process: (self-)counter-feeding (CF) and (self-)counter-bleeding (CB). In CF, ϕ is not sensitive to its own output: in the ISL_2 case, this is exactly when ϕ changes an underlying τ to ρ following ρ in the input. Since ρ and τ are distinct, there can be instances in the output where a surface ρ derived from an underlying τ is followed by a τ in the output. Therefore, opacity is CF if and only if the triggering segment in the environment set A for the non-UR allomorph is also observed in the elsewhere environment set B. In CB, the opposite happens: ϕ destroys its own triggering environment. Specifically for the ISL_2 case, ϕ changes τ to ρ following another input τ. In this case, there may be instances where an underlying τ appears to have changed to ρ following another ρ—exactly in that case when the 'triggering' ρ is derived from an underlying τ. Thus a member of the elsewhere environment set may appear in the triggering environment set.

Thus, algorithmically, opacity can be detected when the intersection between A and B is non-empty: $A \cap B \neq \varnothing$. Specifically, for CF, the true underlying environment sets are A and $B - (A \cap B)$, respectively. For CB, the true underlying environment sets are $A - (A \cap B) \neq \varnothing$ and B, respectively. If the opacity is CF, A contains only triggering segment(s); if the opacity is CB, A also contains one other member of B. In the example, since $A = \{d\}$ and $B = \{t, a, d\}$, the process involves CF opacity because $A \cap B = \{d\} \neq \varnothing$. There is also CB opacity. The reader can confirm that if the rule were $t \rightarrow d \,/\, t \_\_\_$ there may be datasets where $A = \{t, d\}$ and $B = \{a, d\}$. In such a case, the algorithm identifies the triggering environment e as the single member d of the set $A - (A \cap B)$. In general, the triggering segment e is either the only segment of A or $A - (A \cap B)$.

After learning the environment of the process, in the **sixth** step the algorithm deduces the target and the change by the differences in the two allomorphs. Knowing the UR and non-UR SR of a morpheme, the target and change can be easily learned by comparing the mismatch between the two allomorphs. The faithful allomorph w_B begins with the target segment τ, and the modified allomorph w_A exhibits the structural change ρ. The remainder u of w_B is thus the maximal final sequence shared between w_A and w_B. Thus in line 16 the algorithm removes u from the end of w_A and w_B to calculate the structural change ρ. In the example, the two allomorphs are $w_A = da$ and $w_B = ta$. The target is thus the first segment t in ta, the shared final sequence u is a, and so the structural change is da with a removed, or d.

To reflect this change in its hypothesis for the phonology, in line 17 the algorithm changes the output of the transition on τ from state e in T_ϕ to ρ. This creates a transducer that thus implements the rule $\tau \rightarrow \rho \,/\, e \_\_\_$. In the running example, the transition on t on state d is modified such that its output is d, as shown in Fig. 4. The FST T_ϕ will thus change any input t to a d following a d in the input.

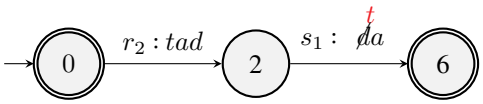


Figure 3: Partial modified T_ℓ

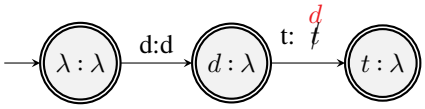


Figure 4: Partial modified T_ϕ.

The **seventh** step and the **final** step of the algorithm, shown in the *for* loop from lines 18 through 25, is thus to update its hypothesis for ℓ. For every morpheme m that has multiple outputs in T_ℓ, the algorithm can identify the UR by finding the output w_B that follows an elsewhere environment $b \in B - e$.

The **final** step is to infer the UR for morphemes incurring opacity, updating their URs. The algorithm searches for such morphemes m' incurring opacity by whether the morpheme ends in α, whether α is the triggering segment, and whether the following morpheme m surface as its UR or not. When the opacity is CF, m' ends in α and $\alpha = e$. m' needs to undo the phonology if the following morpheme m does not have phonology applied in it. When the opacity is CB, m' ends in α but α is not the triggering segment. m' needs to undo the phonology when the following morpheme m does have phonology applied. In this example, based on steps 5, the process involved CF. r_4 with the SR $tadd$ is the morpheme since $T_\ell(r_4)^{-1} T_\ell(r_4 s_1) = ta$ = UR while $T_\ell(r_4) = tadd$ which ends in d, the only element in $\{t, a, d\} \cap \{d\}$. The algorithm computes its UR by undo the phonology ϕ on $tadd$: changing the segment which is mistaken as a triggering segment d to its UR t: $tadd \rightarrow tadt$.[3] The update in the PTT is partially presented in Fig. 5:

[3] This is computed as $tadd(d)^{-1} \cdot t = tadt$, where $\cdot$ is concatenation.

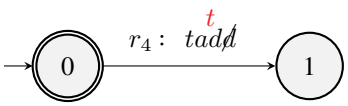


Figure 5: Partial modified T_ℓ

After going through all the steps above, the algorithm finishes the learning process. We can check whether the learning is correct by testing some example data points, and they indeed conform to the observed data set. For example, $\phi \circ \ell(r_2 s_1) = \phi(tad \cdot ta) = tadda$ and $\phi \circ \ell(r_4 s_1) = \phi(tadt \cdot ta) = taddta$.

5. Discussion

We have thus demonstrated with a simple progressive assimilation case how a restrictive ISL_2 characterization of phonology allows for a procedure that can discover URs and a phonology—including non-surface true cases—from morphophonological alternations. The thrust of the algorithm was to take advantage of the fact that the ISL_2 class gives the learner specific hypotheses about a process's environments. This same procedure can be shown to learn epenthesis, deletion, and regressive assimilation and dissimilation. While there are several simplifying assumptions, the goal of this paper has been to demonstrate how classes of functions that make specific hypotheses about the nature of phonological environments provide a way for a learner to navigate the difficult problem of learning URs and a phonology.

Furthermore, this general learning paradigm is computationally efficient and thus cognitively plausible, can be studied analytically so its behavior in the general case is well-understood, and is specific to phonology. We argue that these characteristics make the learner compare favorably to other efforts in learning URs. For example, under Optimality Theory, some systems are computationally difficult. Jarosz (2006) used the idea of maximum likelihood to learn grammars as well as URs which exhaustively evaluates the space of all possible rankings of constraints and the space of possible URs. Apoussidou (2007) designed an algorithm purely based on the interaction of constraints: not only markedness and faithfulness constraints but also lexical constraints that penalize the use of the different possible URs for a morpheme monolithically. These two systems face exceptional computational difficulty since the combinatorics of constraints grow very fast, especially Apoussidou (2007)'s system which explodes the number of constraints with lexical constraints.

Using probabilistic finite state machines, Cotterell et al. (2015) provide a procedure for learning URs and phonology using loopy belief propagation in a directed graphical model whose variables are unknown strings and whose conditional distributions are encoded as finite-state machines with trainable weights. It updates the distribution towards the observed SRs by iteratively updating the parameters. Cotterell's algorithm uses probabilistic finite-state transducer because exact inference based on this framework with string-based variables is uncomputable.

Using the principle of Minimum Description Length (MDL) (Solomonoff, 1964; Rissanen, 1978), Rasin et al. (2015) shows how the simultaneous induction of lexicon, morphological segmentation and phonology is achieved. This is a powerful learning algorithm they demonstrate can learn optionality, opacity and abstract URs (Rasin et al., 2018; Rasin & Katzir). However, it has only been tested on toy data sets, and so its applicability in more general settings is not well understood.

There are three advantages of the proposed algorithm compared with algorithms above. First, since the algorithm imposes a computational structure on possible phonological grammars, it reduces the search space by restricting the possibilities within a region of subregular functions. Therefore it can be computed efficiently and thus is cognitively plausible. Second, this structure encodes a hypothesis specific to phonology and thus makes predictions specific to phonology. Third, this algorithm is provably correct, *i.e.* it can be analyzed as that a theoretical guarantee can be provided about what can be learned and what cannot. Therefore we can understand precisely how the algorithm will behave in any situation.

Of course, there are many simplifying assumptions, most notably that the phonology is ISL_2, and that the data exhibits a single process making a change to a single segment. However, as the learning procedure here is based on the more general idea of modifying the outputs of transitions, it can be used as a first step towards algorithms that can learn other subclasses of regular functions that are defined by a shared structure. This includes the full ISL class, which includes processes in which more than one segment is affected and interactions between processes (see Chandlee et al., 2018). Another such class is the *output* strictly local class (Chandlee, 2014), which can capture iterative spreading processes such as nasal spreading in Johore Malay (Onn, 1980). The output strictly local functions can be learned via a

procedure that assumes shared states but not transitions (Chandlee et al., 2015). The algorithm's ability to discover shared environments in which changes occur may also be able to used to discover natural classes, especially when using finite-state machines that represent features (Heinz & Koirala, 2010).

6. Conclusion

Inferring URs and phonological processes from observed SRs has been both a central problem in understanding how children learn phonology as well as in phonological analysis. This paper aims at proposing a partial solution to this complicated problem. The main theoretical claim of this paper is that knowing the computational property of phonology *a priori* provides avenue for learning URs and a grammar. Assuming that phonology is sub-regular, learning phonology and URs can be demonstrated as a feasible and tractable problem. This line of research can also determine which set of phonological transformations is learnable where strong typological predictions can be made, thereby identifying the universal computational property of phonology from the perspective of learnability. Although this algorithm is very restricted on the type of the phonological transformation and the number of processes that can be learned simultaneously, we expect this work will be a first step towards provably-correct algorithms learning more general and complex phonological transformations.

References

Apoussidou, D (2007). The learnability of metrical phonology: Lot. *Universiteit van Amsterdam [Host]* .

Chandlee, Jane (2014). *Strictly local phonological processes*. Ph.D. thesis, University of Delaware.

Chandlee, Jane & Jeffrey Heinz (2018). Strictly locality and phonological maps. *LI* 49, 23–60.

Chandlee, Jane, Rémi Eyraud & Jeffrey Heinz (2014). Learning Strictly Local subsequential functions. *Transactions of the Association for Computational Linguistics* 2, 491–503.

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

Chandlee, Jane, Jeffrey Heinz & Adam Jardine (2018). Input strictly local opaque maps. *Phonology* 35:2, 171–205.

Cotterell, Ryan, Nanyun Peng & Jason Eisner (2015). Modeling word forms using latent underlying morphs and phonology. *Transactions of the Association for Computational Linguistics* 3, 433–447.

Hayes, Bruce & Colin Wilson (2008). A maximum entropy model of phonotactics and phonotactic learning. *Linguistic inquiry* 39:3, 379–440.

Heinz, Jeffrey (2010). Learning long-distance phonotactics. *Linguistic Inquiry* 41:4, 623–661.

Heinz, Jeff (2018). The computational nature of phonological generalizations. *Phonological Typology, Phonetics and Phonology* p. 69.

Heinz, Jeffrey & Cesar Koirala (2010). Maximum likelihood estimation of feature-based distributions. *Proceedings of the 11th Meeting of the ACL Special Interest Group on Computational Morphology and Phonology*, Association for Computational Linguistics, Uppsala, Sweden, 28–37.

Jardine, Adam, Jane Chandlee, Rémi Eyraud & Jeffrey Heinz (2014). Very efficient learning of structured classes of subsequential functions from positive data. *International Conference on Grammatical Inference*, 94–108.

Jarosz, Gaja (2006). *Rich lexicons and restrictive grammars: Maximum likelihood learning in Optimality Theory*. Ph.D. thesis, Johns Hopkins University.

McCarthy, John J (1999). Sympathy and phonological opacity. *Phonology* 16:3, 331–399.

Mielke, Jeff (2004). P-base: Database of sound patterns.

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

Onn, Farid Mohd (1980). *Aspects of Malay phonology and morphology: A generative approach*. Universiti Kebangsaan Malaysia.

Rasin, Ezer & Roni Katzir (). Learning abstract underlying representations from distributional evidence .

Rasin, Ezer, Iddo Berger & Roni Katzir (2015). Learning rule-based morpho-phonology. *Work. pap., MIT, Cambridge, MA Google Scholar Article Location* .

Rasin, Ezer, Iddo Berger, Nur Lan & Roni Katzir (2018). Learning phonological optionality and opacity from distributional evidence. *Proceedings of NELS*, vol. 48.

Rissanen, Jorma (1978). Modeling by shortest data description. *Automatica* 14:5, 465–471.

Solomonoff, Ray J (1964). A formal theory of inductive inference. part i. *Information and control* 7:1, 1–22.

Tesar, Bruce (2014). *Output-driven phonology: Theory and learning*. 139, Cambridge University Press.