Constraint-based Learning of Phonological Processes

Shraddha Barke, Rose Kunkel, Nadia Polikarpova, Eric Meinhardt, Eric Baković, and Leon Bergen

{sbarke,wkunkel,npolikarpova,emeinhar,ebakovic,lbergen} @ucsd.edu

University of California, San Diego

Abstract

Phonological processes are context-dependent sound changes in natural languages. We present an unsupervised approach to learning human-readable descriptions of phonological processes from collections of related utterances. Our approach builds upon a technique from the programming languages community called *constraint-based program synthesis*. We contribute a novel encoding of the learning problem into Satisfiablity Modulo Theory constraints, which enables both data efficiency and fast inference. We evaluate our system on textbook phonology problems and lexical databases, and show that it achieves high accuracy at speeds two orders of magnitude faster than state-of-the-art approaches.

1 Introduction

Phonological processes govern the way speech sounds in natural languages change depending on the context. For example, in English verbs, the past tense suffix /d/ turns into [t] after voiceless consonants (so the word "zipped" is pronounced [zɪp**t**], while "begged" is pronounced [bɛg**d**]). Linguists routinely face the task of inferring phonological processes by observing and contrasting *surface forms* (pronunciations) of morphologically related words. To aid linguists with this task, we consider the problem of learning phonological processes automatically from collections of related surface forms.

This problem setting imposes four core requirements, which guide the design of our approach:

1. Inference results must be **fully interpretable**: our goal is to *explain* phonological processes exhibited by the data, not merely *predict* pronunciations of unseen words. Hence, our model takes the form of discrete, conditional rewrite rules from rule-based phonology (Chomsky and Halle, 1968).

2. Inference must be **unsupervised**: phonological processes are formalized as transformations from (latent) *underlying forms* to surface forms (rather than *between* surface forms).

3. Inference must be **data-efficient**: typically only a handful of data points are available.

4. Inference must be **fast**: we envision linguists using our system interactively, tweaking the data and being able to see the inferred rules within minutes.

Recently *program synthesis* has emerged as a promising approach to interpretable and data-efficient learning (Ellis et al., 2015; Singh et al., 2017; Verma et al., 2018; Ellis et al., 2018). In program synthesis, models are represented as programs in a domain-specific language (DSL), which allows domain experts to impose a strong prior by designing an appropriate DSL. Program synthesis uses powerful constraint solvers to perform combinatorial optimization and find the least-cost program in the DSL that fits the data. Program synthesis has been previously used to tackle the problem of phonological rule learning (Ellis et al., 2015), however their work uses global inference which scales poorly and hence does not satisfy requirement 4 (their system takes an hour on average to solve a phonology textbook problem).

In this work, we propose a novel inference technique that satisfies all four core requirements. Our key insight is that the problem of learning conditional rewrite rules can be decomposed into three steps: inference of the latent *underlying forms*, learning the *changes* (rewrites), and learning the *conditions*. Moreover, each of these problems can be encoded as a constrained optimization problem that can be solved efficiently by modern *satisfiability modulo theories* (SMT) solvers (de Moura and Bjørner, 2008). Both the decomposition and the encoding into constraints are contributions of this work. We implement this

approach in a system called SYPHON and show that it is capable of generating accurate phonological rules in under a minute and from just 5–30 data points.

2 Background and Problem Definition

In this section, we illustrate phonological processes and the problem of phonological rule induction using our running example of English verbs.

2.1 Rule-Based Phonology

Phonological features. *Phones* (speech sounds) are described using a feature system that groups similar-sounding phones together. For instance, voiced consonants (consonants produced with vibrating vocal cords, like [z], [d], [b]) possess the features +consonant and +voice, while voiceless consonants (like [s], [t], [p]) possess the features +consonant and −voice. Each phone can be uniquely identified by a feature vector: for example [−voice −strident +anterior −distributed] uniquely identifies the sound [t]. However, some phones may be uniquely identified by several feature vectors, and not all feature vectors correspond to phones (the feature system is redundant). For example, the feature vector [+low +high] does not correspond to any phones, as no phone can have both a raised and a lowered tongue body.

Phonological rules. In rule-based phonology, a phonological process is formalized as a conditional rewrite rule that transforms an *underlying form* of a word (roughly, the unique stored form of the word) into its *surface form* (the word as it is intended to be pronounced). In our English past tense example, the underlying form /zɪpd/—formed by concatenating the stem /zɪp/ and past tense suffix /d/—is transformed into the surface form [zɪpt] by a rule that makes an obstruent voiceless when it occurs after a voiceless obstruent:

$$[-\text{sonorant}] \rightarrow [-\text{voice}] \,/\, [-\text{voice}] \_$$

In general, phonological rules have the form $A \rightarrow B \;/\; L \_ R$, where all of A, B, L, and R are feature vectors. The rule means that any phone that matches A and occurs between two phones that match L and R, respectively, will be rewritten to match B (leaving the features not mentioned by B intact). A is called the *target* of the rule, B is called the *structural change*, and L and R are the left and the right *contexts*.[1] In the example above, the right context is omitted, because it is irrelevant to the rule's application; formally, A, L, and R may each be empty feature vectors, which are defined to match any phone.

Hereafter, we refer to the sequence LAR of the target and the context as the *condition* of the rule. If the condition is empty, the rule applies unconditionally. In addition to + and −, the values of features in the condition of the rule may be *variables*, which enforce that features have the same value in different parts of the condition. For example,

$$A \rightarrow B \,/\, [\alpha\text{consonant}] \_ [\alpha\text{consonant}]$$

describes a rule which applies between pairs of consonants and pairs of vowels, but not between a consonant and a vowel.

2.2 Problem Definition

The input to our problem is a matrix of surface forms, such as the one shown in Fig. 1, left. These forms are arranged into rows, corresponding to different stems, and columns, corresponding to different inflections (in this case, the third-person singular and past tense of English verbs). In the interest of space, we only show four rows from this data set, but a typical input in a phonology textbook problem is only slightly larger and ranges from 5 to 30 rows.

Given these data, our task is to infer the latent underlying forms for each of the words in the input such that the resulting matrix of underlying forms factorizes into stems and suffixes, and to learn a sequence of phonological rules which, when applied elementwise to the matrix of underlying forms, reproduces the matrix of surface forms.

This learned sequence of phonological rules is generative in the following sense: given the underlying form for a new word, such as /æskz/, we can deterministically apply these rules to generate the surface form of that word, [æsks]. We use this property to evaluate the accuracy of the rule set we learned by holding out a portion of the words from the data, and then applying the rule to the underlying forms of those words, which were determined through phonological research.

2.3 Phonological Intuition

The design of our system is informed by how linguists solve the problem of phonological rule induction. When a phonologist analyzes these data, they begin by positing underlying forms that are likely to result in the simplest set of rules. For example, they observe that the substring shared in each row is most likely the stem, which surfaces without change; the underlying suffix in the first column in Fig. 1 is likely

[1] In this work we only consider a subset of *strictly local* k=3 rules (Chandlee et al., 2014), where either side of the context is restricted to at most a single phone.

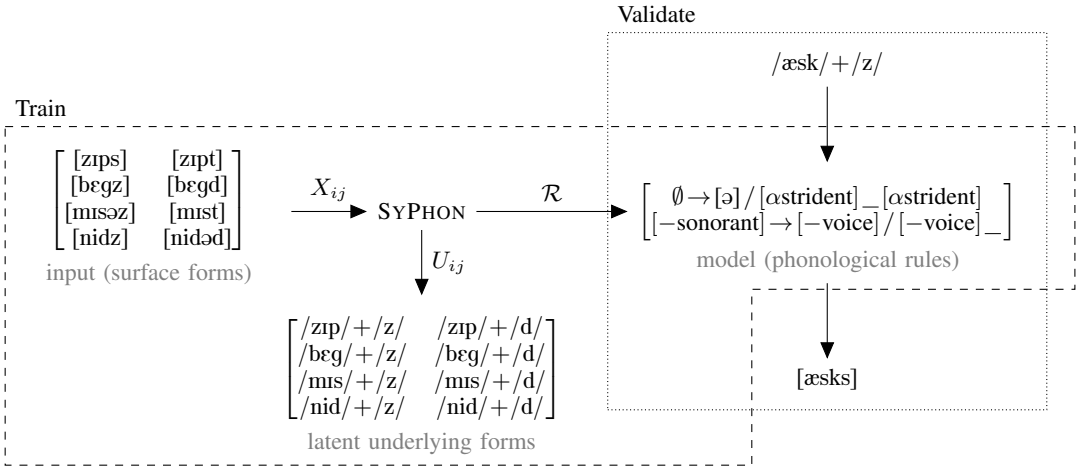

Figure 1: The general structure of the problem, shown concretely for English verbs.

/z/, which sometimes surfaces as [s] and other times as [əz]; and similarly, the underlying suffix in the second column is likely /d/, which can change to [t] or [əd]. The choice of /z/ and /d/ as the underlying suffixes is preferred to, say, /s/ and /t/, because this choice lets us explain all the observed data using only three edits: /z/ → [s], /d/ → [t], and ∅ → [ə].

The next step is to merge and generalize individual edits: the first two edits are both devoicing an obstruent, so they can be merged into [−sonorant] → [−voice], while the last edit is an insertion and cannot be generalized.

The final step of the analysis is to infer the conditions under which each of the two structural changes occurs. By contrasting examples in the first column, we infer that the insertion happens when the suffix /z/ occurs after a strident (like /s/ in /mɪs/); otherwise, /z/ and /d/ are devoiced whenever they occur after a voiceless obstruent (like /p/ in /zɪp/). The full data set can be explained using the two rules in Fig. 1, right. Note that in order to capture the data in both columns, the insertion rule says that [ə] is inserted whenever the stridency of the left and right context matches. Note also that in this case the order of rules matters: for words like /mɪsz/, insertion is applied first, which prevents the devoicing rule from applying.

3 Learning Phonological Rules

As illustrated in Fig. 1, the input to our learning problem is a matrix of surface forms X_{ij} with I rows and J columns. The goal is to learn a discrete rule set $\mathcal{R}$, while jointly inferring the latent set of I stems S_i and J affixes A_j.

Hypothesis space. The hypothesis space for $\mathcal{R}$ can be formalized as a context-free grammar:

$$\begin{aligned} &\mathcal{R} \Rightarrow R^* && R \Rightarrow C \rightarrow C \,/\, C\_C \\ &C \Rightarrow (VF)^* && V \Rightarrow + \,|\, - \\ & && F \Rightarrow \text{consonant} \,|\, \text{voice} \,|\, \ldots \end{aligned} \tag{1}$$

According to this grammar, $\mathcal{R}$ is a sequence of rules R; each R is defined in terms of four feature vectors C; each feature vector is a sequence of pairs of feature values V and feature names F.

Rewriting. We use $\mathcal{C}_R$ and $\mathcal{B}_R$ to denote the condition and structural change of a rule R, respectively. A feature vector C can be interpreted as a Boolean formula that holds of a phone a if a possesses all features in C; we denote by $|C|$ the number of models of this formula, *i.e.* phones in the inventory Φ that satisfy C. Similarly, $\mathcal{C}_R$ is a Boolean formula over *trigrams* of phones. A *rewrite* of a trigram abc by rule R is defined as:

$$R(abc) = \begin{cases} \mathcal{B}_R(b) & \text{if } \mathcal{C}_R(abc) \\ b & \text{otherwise} \end{cases}$$

The notion of rewrites can be extended to words and rule sets.

Learning as constrained optimization. We can now formalize our problem as a hard *correctness constraint* over rules and underlying forms U_{ij}:

$$\mathcal{R}(U_{ij}) = X_{ij} \quad \text{where } U_{ij} \triangleq A_j[S_i] \tag{2}$$

Here, $A_j[S_i]$ denotes a concatenation of the prefix/suffix A_j with the stem S_i.

There might be many rule sets $\mathcal{R}$ that are consistent with all the data, and what we would like is to pick one that generalizes to other data that exhibits the same phonological process (for example, the rule inferred in Fig. 1 should generalize to other regular English verbs). Hence we frame the learning problem

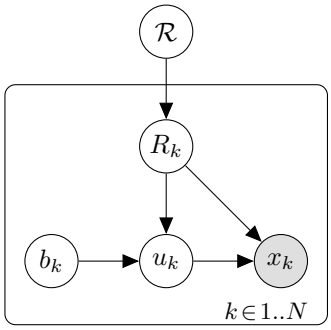

Figure 2: Probabilistic model of a phonological process. A rule set $\mathcal{R}$ is sampled from a description length prior. We observe a set of N surface phonemes x_k; each x_k is generated by sampling a rule R_k from $\mathcal{R}$ and an underlying trigram u_k, and deterministically applying R_k to u_k (coin flip b_k decides whether u_k should match R_k's condition).

as a constrained optimization problem and derive the objective function using a Bayesian model.

3.1 Bayesian Model

Generative process. Intuitively, to generate surface forms X_{ij}, we must sample a single rule set $\mathcal{R}$, I stems S_i, and J affixes A_j, and then deterministically apply $\mathcal{R}$ to each $A_j[S_i]$. Prior work on phonological rule learning (Ellis et al., 2015) assumed that S_i and A_j are sampled uniformly from the language and independently of $\mathcal{R}$. We observe, however, that in most data sets of interest, underlying forms are in fact sampled to contrast the contexts in which $\mathcal{R}$ does and does not apply. We model this intuition as a *strong sampling* process depicted in Fig. 2.

For simplicity, in this model each observation corresponds to an individual rule application to an underlying trigram u that produces a surface phoneme x. For example, the rewrite /zɪpz/ → [zɪps] is represented as four observations: /#zi/ → [z], /zɪp/ → [ɪ], /ipz/ → [p], and /pz#/ → [s] (where # encodes word boundary).

Our generative process first samples a ruleset $\mathcal{R}$ from the description length prior over the hypothesis space (1):

$$P(\mathcal{R}) \propto 2^{-w_s \cdot \sum_{R \in \mathcal{R}} \ell(R)}$$

where $\ell(R)$ is the length of rule R and $w_s > 0$ is a model hyperparameter. For each observation $k \in 1..N$, we pick a rule R_k uniformly from $\mathcal{R}$. Before sampling the underlying trigram u_k, we flip a coin b_k to decide whether we want to sample a *positive* or a *negative* trigram, *i.e.* whether $\mathcal{C}_{R_k}(u_k)$ should hold true; we then sample u_k uniformly from the set of all positive (resp. negative) trigrams (subject to the hard constraint that they form a factorizable matrix U_{ij}). Finally, we deterministically compute $x_k \triangleq R_k(u_k)$. Hence we can define:

$$P(x_k, u_k \mid R_k) = \begin{cases} 0 & \text{if } R_k(u_k) \neq x_k \\ \frac{P(b_k = \top)}{|\mathcal{C}_{R_k}|} & \text{if } \mathcal{C}_{R_k}(u_k) \\ \frac{P(b_k = \bot)}{|\neg\mathcal{C}_{R_k}|} & \text{otherwise} \end{cases}$$

Our goal is to maximize

$$P(\mathcal{R}, R_1, ..., R_N, u_1, ..., u_N \mid x_1, ..., x_N)$$
$$\propto P(\mathcal{R}) \prod_{k=1}^{N} P(x_k, u_k \mid R_k) P(R_k \mid \mathcal{R})$$

Objective function. Taking logs, we can derive the following approximate minimization objective for our constrained optimization problem:

$$w_s \sum_{R \in \mathcal{R}} \ell(R) + N_R^+ \cdot \log(|\mathcal{C}_R|) \qquad (3)$$

where N_R^+ is the number of positive examples for this rule. (Note that this objective ignores $P(R_k \mid \mathcal{R})$ and b_k, which are assumed to be uniform. It also ignores the negative examples. This provides a reasonable approximation, under the assumption that $|\neg\mathcal{C}_R| \gg |\mathcal{C}_R|$ for each rule R, which holds in the current setting.) This function includes a *simplicity term*, which favors rules with shorter (and hence, more general) conditions, and a *likelihood term*, which favors more specific conditions if there are sufficient positive examples to support them. This likelihood term stems from our strong sampling assumption; we demonstrate its importance for inferring accurate rules in Sec. 5.

3.2 Inference by Program Synthesis

To solve the constrained optimization problem we build upon a technique from programming languages called *constraint-based program synthesis* (Solar-Lezama, 2013).

Constraint-based synthesis. The input to (inductive) program synthesis is a DSL that defines the space of possible programs and a set of input-output examples $E = \overrightarrow{\langle i, o \rangle}$; the goal is to find a program whose behavior is consistent with the examples. In constraint-based synthesis, this search problem is reduced to solving a boolean constraint. To this end, we index the DSL by a bitvector c, called a *control vector*. We then define a mapping from control vectors to program behaviors via an *evaluation relation* $\varphi(c, y, z)$—a boolean formula that holds if and only if a program indexed by c produces output z on input y. Given the evaluation relation, the synthesis problem reduces to solving the following boolean constraint:

$$\exists c. \bigwedge_{\langle i, o \rangle \in E} \varphi(c, i, o)$$

An SMT solver (de Moura and Bjørner, 2008) is then used to find a satisfying assignment for c, which allows us to recover the corresponding program. For this approach to succeed, the evaluation relation has to be designed carefully so that it only uses constraints that the solver can efficiently reason about.

Synthesis of phonological rules. In our setting, the DSL is the space of all rule sets $\mathcal{R}$ (up to a certain size), and the evaluation relation $\varphi(c,U,X)$ is the correctness condition (2). Importantly, our setting differs from traditional program synthesis in two ways: first, we have to search for both the control vector and the inputs, and second, in addition to satisfying the correctness condition, we also seek to minimize the objective function (3). If we encode the objective function as $\psi(c,\overrightarrow{\langle U,X\rangle})$, we can reduce rule learning to the following constrained optimization:

$$\textbf{minimize} \quad \psi(c,\overrightarrow{\langle A_j[S_i],X_{ij}\rangle})$$

$$\textbf{subject to} \quad \bigwedge_{i,j=1,1}^{N,M} \varphi(c,A_j[S_i],X_{ij})$$

Given a proper encoding of φ and ψ, this constraint can be solved by an optimizing SMT solver (Bjørner et al., 2015); this is the approach used in prior work (Ellis et al., 2015). However, this is a very computationally intensive problem. The reason is the astronomical size of the search space: for a problem of factorizing a 10×2 matrix X_{ij} into stems of length $\ell_S = 3$ and affixes of length $\ell_A = 2$, if we limit the maximum number of rules $N_{\mathcal{R}}$ to 2 and consider an inventory Φ with 90 phones and a feature set F with 30 features, we can estimate the size of the search space as $3^{|F|N_{\mathcal{R}}}|\Phi|^{I\ell_S+J\ell_A} \approx 2^{600}$.

Decomposition. To achieve scalable inference, we decompose the global constrained optimization problem into three steps, inspired by phonological intuition we described in Sec. 2.3:

1. *Underlying form inference.* In the first step we use an SMT solver to generate likely underlying stems and suffixes. We rank them based on the heuristic that underlying forms U_{ij} that have a smaller edit distance from surface forms X_{ij} are more likely to produce simple rules (Sec. 3.3).

2. *Change inference.* Given the set of edits between each U_{ij} and the corresponding X_{ij}, we identify the smallest set of structural changes B that can describe all the edits (Sec. 3.4).

3. *Condition inference.* Finally, for each structural change B, we use program synthesis to infer the condition under which this change occurs (Sec. 3.5). If this step fails, we go back to step 1 and generate the next candidate matrix U_{ij}.

In the rest of this section we detail these three steps. For illustration purposes, in all examples we will assume that our feature set has just three features: voice v, sonorant s, and continuant c.

3.3 Underlying Form Inference

The input to this step is the matrix of surface forms X_{ij} and the output is a set of aligned pairs $\langle U,X\rangle_{ij}$. Tab. 1 illustrates this for a 2×2 matrix of English verbs. For example, $\langle U,X\rangle_{11} = \langle$[zɪpz],[zɪps]$\rangle$; we use red to show alignment information (in this case, a single substitution $z \to s$). Insertions and deletions are represented by alignment with null segments.

Input		Output	
[zɪps]	[zɪpt]	[zɪpz] [zɪps]	[zɪpd] [zɪpt]
[nidz]	[nidəd]	[nidz] [nidz]	[nid_d] [nidəd]

Table 1: Underlying form inference on English verbs.

The output matrix $\langle U, X\rangle_{ij}$ has to satisfy two properties: (i) the matrix U_{ij} can be factorized into stems S_i and affixes A_j, and (ii) each pair $\langle U,X\rangle$ has a small edit distance. Our intuition is that underlying forms that have a small edit distance from surface forms are likely to produce simpler rules. Hence we generate candidate matrices $\langle U,X\rangle_{ij}$ in the order of increasing edit distance, until rule inference succeeds for one of them. This strategy will always eventually find a matrix of underlying forms which can be related to the surface forms by a rule set we can infer as long as one exists. This process is not guaranteed to find the global minimum of the objective function (3), but we show empirically that it produces good results.

We can encode the properties (i) and (ii) as a boolean constraint over unknown strings with concatenation and length, which can be solved efficiently by the Z3STR2 solver (Zheng et al., 2017). From the solutions for those unknowns it is straightforward to recover not only the stems and suffixes, but also the required alignment information between the underlying and surface forms.

3.4 Change Inference

The input to change inference is the set of all edits in the aligned pairs $\langle U,X\rangle_{ij}$, computed in the previous step, and the output is a set of structural changes

that captures all the edits. Tab. 2 illustrates this for the edits from Tab. 1; columns LHS and RHS show relevant features of the left- and and right-hand sides of the edit.

Edit	LHS	RHS	Change
/z/ → [s]	[+v −s +c]	[−v −s +c]	[−v]
/d/ → [t]	[+v −s −c]	[−v −s −c]	
∅ → [ə]	∅	[ə]	[ə]

Table 2: Change inference on English verbs.

For each edit, we compute the set of all possible structural changes which are consistent with the edit. For example, the edit /z/ → [s] is consistent with four possible changes: [−v], [−v −s] [−v +c], and [−v −s +c]. Next, we greedily merge change-sets of different edits if their intersection is nonempty. This merging step allows us to identify a small set of distinct structural changes which together describe all the edits. For example, the change-sets of the first two edits in Tab. 2 can be merged to produce the change-set: {[−v],[−v −s]}. The third edit in Tab. 2 is an insertion, which changes the values of *all* features present in [ə], and hence cannot be merged. When no more merges are possible, we pick the simplest change from every change set (in this case, we end up with changes $B_1 = [-\text{v}]$ and $B_2 = [\text{ə}]$). This greedy process bounds the maximum number of rules to the number of change sets.

3.5 Condition Inference

For each structural change B inferred in the previous step, we now attempt to determine the condition LAR under which the change applies. If successful, a rule $A \rightarrow B \;/\; L \_ R$ is added to the inferred rule set $\mathcal{R}$; otherwise we go back to underlying form inference and try the next candidate matrix U_{ij}.

For a given change B, the input to condition inference is the set of pairs $\langle u, \ell \rangle_k$, where u_k is a phone trigram in some underlying form and the label ℓ_k can be *positive* ($\top$), *negative* ($\bot$), or *unknown* (?). Tab. 3 illustrates this for trigrams from $U = /\text{zɪpz}/$. A trigram is labeled positive if its middle phone undergoes the change B in the data, negative if it does not undergo B, and unknown if B has no effect on this phone. In our example, neither /ɪ/ nor /p/ in /zɪps/ actually changed, however /zɪp/ is labeled $\bot$ while /ɪpz/ is labeled ?, because /p/ is already [−v], and hence devoicing has no effect on it. Our goal is to infer a condition consistent with the labels of all the positive and negative trigrams (unknown trigrams are ignored).

u	ℓ	Features		
/#zɪ/	$\bot$	[+#]	[+v −s]	[+v +s]
/zɪp/	$\bot$	[+v −s]	[+v +s]	[−v −s]
/ɪpz/	?	[+v +s]	[−v −s]	[+v −s]
/pz#/	$\top$	[−v −s]	[+v −s]	[+#]

Table 3: Input to condition inference for change [−v] on /zɪpz/ → [zɪps]

Inference by program synthesis. To frame condition inference as a program synthesis problem we need to define the control vector that indexes the space of all possible conditions, and a corresponding evaluation relation. In our control vector, for each feature f, we use six control variables, which represent the three positions that a feature can appear in a rule (left context, target, and right context) and the two values it can take on (+ and −). We denote these variables by f_p^v for v in $V = \{+,-\}$ and p in $P = \{l,t,r\}$.

Our evaluation relation takes the form $\varphi(c,u,\ell) \triangleq \text{matches}(c,u) \Leftrightarrow \ell$, where matches is a relation specifying whether the condition indexed by c matches the trigram u. The matches relation is further defined as follows:

$$\text{matches}(c,u) \triangleq \bigwedge_{(f,v,p)\in F\times V\times P} f_p^v \Rightarrow (u_{p,f} = v)$$

where $u_{p,f}$ is the value of feature f at position p in trigram u.

3.6 Inductive Bias

In addition to being consistent with the data, we also want the condition to minimize the objective function (3). We encode the objective function as

$$w_s s(c) + l(c),$$

where $s(c)$ encodes the simplicity of the condition indexed c (its size), $l(c)$ encodes the likelihood, and w_s is a model hyperparameter which determines the relative importance of simplicity.

The challenge is to encode the likelihood term in a solver-friendly way. To count the number of models of $|\mathcal{C}_R|$, we observe that $|\mathcal{C}_R| = |\mathcal{C}_R^l||\mathcal{C}_R^t||\mathcal{C}_R^r|$, *i.e.* we can independently count the models of the target, and the left and right contexts, so

$$l(c) \triangleq N^+ \sum_{p\in P} \log(|\mathcal{C}_R^p|)$$

We also observe that $|\mathcal{C}_R^p|$ can be encoded efficiently using a constraint whose size is *linear* in the size of the phone inventory Φ:

$$|\mathcal{C}_R^p| \triangleq \sum_{a\in\Phi} \text{if} \Big(\bigwedge_{\substack{(f,v)\in F\times V \\ a_f \neq v}} \neg f_p^v \Big) \text{ then } 1 \text{ else } 0$$

Finally, as the solver does not support logarithms, we encode log using a lookup table. This is tractable, since we only need to evaluate the log of each $|\mathcal{C}_R^p|$, which is at most the size of our inventory, roughly 100 phones.

3.7 Current limitations

SYPHON currently leverages three simplifying assumptions about rules for domain-specific problem decomposition and SMT encoding, which are crucial to making learning computationally tractable.

Conjunctive conditions. Rule conditions are conjunctions of equalities over feature values, and each rule has a unique change. We can thus decompose the learning process into change inference and condition inference: change inference greedily groups all observed edits into changes, and from then on we assume that each change uniquely corresponds to a rule.

Local context. The condition of each rule is only a function of the target phone and two surrounding phones. This allows us to encode condition inference as learning a formula over *trigrams* of phones, which has a compact encoding as SMT constraints.

Rule interaction. One rule's change does not create conditions for another. This allows us to perform condition inference for each rule independently.

Many attested patterns in real languages go beyond these limitations. We believe that it is possible to lift these restrictions, and still leverage the structure of conditional rewrite rules to retain most of the benefits of our problem decomposition. We leave this extension to future work.

4 Data

We evaluate our system on two broad categories of datasets: lexical databases and textbook problems.

4.1 Lexical Databases

We use large lexical databases to investigate two (morpho)phonological processes in English: flapping (6457 rows) and regular verb inflections (2756 rows). We process the CMU pronouncing dictionary (Weide, 2014) to create underlying and surface form pairs exemplifying flapping, as in Gildea and Jurafsky (1996). For verb inflections, we combine morphological information extracted from CELEX-2 (Baayen et al., 1993) with CMU transcriptions to create a database of regular verbs, where each row of the database contains the third-person singular present tense wordform and past tense wordform for a given verb. For both datasets we have gold standard solutions for both rule sets and underlying forms, provided by one of our coauthors, who is a phonologist.

4.2 Textbook Problems

For this category, we curated a set of 34 problems from phonology textbooks (Gussenhoven and Jacobs, 2017; Odden, 2005; Roca and Johnson, 1999) by selecting problems with local, non-interacting rules. For each problem, the input data set is tailored (by the textbook author) to illustrate a particular phonological process, and contains 20-50 surface forms. For all of these problems we have gold standard solutions, either provided with the textbook or inferred by a phonologist. Gold standard solutions range in complexity from one to two rules. Out of the 34, 21 problems are shared with (Ellis et al., 2019), which we use as the baseline for inference times.

Following the textbooks, these problems are further subdivided into 10 *matrix problems*, 20 *alternation problems*, and 4 *supervised problems*. The matrix problems follow the format presented in Sec. 2. The alternation and supervised problems are easier, as we are given more information about the underlying form. For alternation problems, we are essentially given a set of choices for what the underlying form might be, and for supervised problems the underlying form is given exactly. These problems include examples of phones in complementary distribution. Our problem decomposition allows us to switch out underlying forms inference to handle different kinds of input. According to this classification, the flapping lexical database is an alternation problem and verbs is a matrix problem.

5 Experiments

We evaluate our system on the two categories of data sets discussed in Sec. 4. We split the 34 textbook problems into 24 development and 10 test problems. Our system has several free parameters (most importantly, the simplicity weight w_s). These were hand-tuned on all of the data except the test problems. For the test problems, we only added missing sounds to the inventory as needed. The 10 test problems are all alternation problems. We leave for future work the investigation of these hyperparameter settings on new matrix problems.

5.1 Lexical Database Experiments

We evaluate our system on two large English datasets, one demonstrating flapping, and the other verbs. For each dataset, we learn a rule set from 20, 50 and 100

	Accuracy		Rule Match				UF
			Precision		Recall		
	SP	SP-	SP	SP-	SP	SP-	
Flap 20	76	52	50	66	31	25	100
Flap 50	93	79	86	71	86	71	100
Flap 100	100	79	100	71	100	71	100
Verb 20	86	73	48	42	83	61	100
Verb 50	88	78	52	50	92	80	100
Verb 100	95	81	62	58	100	82	100

Table 4: Accuracy results for the English flapping and verbs corpora data sets on 20, 50 and 100 training examples. SYPHON (SP) and SYPHON- (SP-) are two variants of our model, with and without likelihood, resp. Accuracy reports the generalization accuracy on unseen inputs, rule match and UF indicate how well the inferred rule and underlying form resp. match the gold standard.

data points, and evaluate its accuracy on the remaining data. We also perform a syntactic comparison of the rule set against the gold standard rules, which we report as average precision and recall among the sets of features in the two rules. Finally, we compare the latent underlying forms we inferred for each problem to the known correct underlying forms. Tab. 4 summarizes the results. Tab. 5 (rows 1–3) shows the actual rules inferred on the three flapping training sets.

To examine the importance of likelihood in our system, we repeat this experiment for a variant of our system SYPHON-, which does not consider likelihood and simply optimizes our simplicity prior. As the number of data points increases, the effect of the likelihood grows, and so for SYPHON the recall compared to the gold standard quickly climbs. By contrast, the recall of SYPHON- plateaus, which shows that likelihood is indeed important for finding good rules.

5.2 Textbook Problem Experiments

We evaluate the textbook problems under the following three experimental conditions. First, to evaluate the generalization accuracy for unseen inputs, for each of the problems, we hold out a randomly sampled 20% of the data. We then learn a rule set on the remaining data, and validate it against the held out data. We repeat this process three times, and report the average accuracy for each class of problems in Tab. 6. We also evaluate syntactic accuracy of the rules and of underlying forms, in the same way as for the lexical databases. Additionally, we evaluate our system on 10 test problems, which were left out entirely when tuning the system hyperparameters. We report the same metrics for these problems. Tab. 5 shows inferred rules for selected development problems (rows 4–8) and test problems (rows 9–13).

	Data Set	Inferred Rule
1	Flap 20	$\begin{bmatrix}+\text{cor}\\-\text{voi}\end{bmatrix} \rightarrow [+\text{approx}] \,/\, [+1\text{stress}]\_$
2	Flap 50	$\begin{bmatrix}+\text{cor}\\-\text{voi}\\-\text{del. rel.}\end{bmatrix} \rightarrow \begin{bmatrix}+\text{voi}\\+\text{approx}\end{bmatrix} \,/\, [+\text{stress}]\_[+\text{syll}]$
3	Flap 100	$\begin{bmatrix}+\text{ant}\\-\text{voi}\\-\text{del. rel.}\end{bmatrix} \rightarrow \begin{bmatrix}+\text{voi}\\+\text{approx}\end{bmatrix} \,/\, [+\text{stress}]\_[+\text{syll}]$
4	Russian	$[-\text{son}] \rightarrow [-\text{voi}] \,/\, \_\#$
5	Scottish	$[+\text{syll}] \rightarrow [+\text{long}] \,/\, \_ \begin{bmatrix}+\text{cons}\\+\text{voi}\\+\text{cont}\end{bmatrix}$
6	Korean	$\begin{bmatrix}-\text{cont}\\-\text{voi}\end{bmatrix} \rightarrow \begin{bmatrix}-\text{c.g.}\\-\text{s.g.}\end{bmatrix} \,/\, \_[+\text{c.g.}]$
7	Farsi	$\begin{bmatrix}-\text{cont}\\+\text{dors}\end{bmatrix} \rightarrow \emptyset \,/\, [+\text{ATR}]\_\#$
8	Hungarian	$[-\text{son}] \rightarrow [\alpha\text{voi}] \,/\, \_ \begin{bmatrix}\alpha\text{voi}\\-\text{del. rel.}\end{bmatrix}$
9	Kishambaa	$[+\text{nas}] \rightarrow [-\text{voi}] \,/\, \_[+\text{s.g.}]$
10	Persian	$\begin{bmatrix}+\text{approx}\\-\text{voi}\end{bmatrix} \rightarrow [+\text{voi}] \,/\, \_[-\text{nas}]$
11	Ganda	$[+\text{lat}] \rightarrow [+\text{cont}] \,/\, \begin{bmatrix}-\text{lab}\\+\text{ATR}\end{bmatrix} \_$
12	Limbu	$\begin{bmatrix}+\text{back}\\+\text{syll}\end{bmatrix} \rightarrow [+\text{rnd}] \,/\, \begin{bmatrix}+\text{lab}\\-\text{cont}\end{bmatrix} \_$
13	Kongo	$\begin{bmatrix}-\text{son}\\+\text{cor}\end{bmatrix} \rightarrow \begin{bmatrix}-\text{ant}\\+\text{dist}\\+\text{strid}\end{bmatrix} \,/\, \_ \begin{bmatrix}-\text{rnd}\\+\text{high}\end{bmatrix}$

Table 5: Selected inferred rules: English flapping trained on 20, 50 and 100 examples (1–3); textbook development problems (4–8); textbook test problems (9–13).

The accuracy of our inferred rules and underlying forms is 100% for all textbook problems. This is not surprising: the combination of hard constraints and a restrictive DSL makes inferring incorrect rules or underlying forms very difficult. More interesting is the syntactic comparison to the gold standard. This measure is intended to estimate how well the rules SYPHON produces match phonologists' intuition. The results in Tab. 6 confirm that without the likelihood term, inference tends to exclude important features from the rule condition: the recall for held out problems goes down by 21%.

Finally, we compare inference times of SYPHON with the prior work of Ellis et al. (2019), which is also based on constraint-based program synthesis but does not perform problem decomposition, instead using the global encoding outlined in Sec. 3.2. As shown in Tab. 7, the decomposition makes SYPHON at least *two orders of magnitude* faster, with an average inference time of just 30 seconds for matrix problems, thus enabling phonologists to use the tool interactively.

	Accuracy		Rule Match				UF
			Precision		Recall		
	SP	SP-	SP	SP-	SP	SP-	SP
MAT	100	100	70	69	77	69	100
ALT	100	100	66	61	71	62	100
SUP	100	100	63	60	71	64	-
TEST	100	100	54	52	61	48	100

Table 6: Accuracy of textbook problems. We use (-) for supervised problems without underlying form inference.

	Inference Time (secs)		
	SYPHON	Baseline	Speedup
MAT	30.0	3100	124.6
ALT	10.7	N/A	N/A
SUP	5.3	6333	378.3
TEST	8.3	N/A	N/A

Table 7: Comparison of the inference times of textbook problems with baseline. We report the median execution times and geometric mean of the speedups. N/A indicates examples where baseline results are unavailable.

6 Related Work

Learning (morpho-)phonology is a rich and active area of research; in this overview, we focus on approaches that share our goal of inferring fully interpretable models of phonological processes.

Most closely related to ours is the work of Ellis et al. (2015) and its (unpublished) follow-up (Ellis et al., 2019) on using program synthesis to infer phonological rules. As mentioned above, the main difference is that SYPHON is two orders of magnitude faster than their system thanks to a novel decomposition and efficient SMT encoding. On the other hand, we impose extra restrictions on the hypothesis space (*i.e.* we only support local rules), which means that SYPHON is unable to solve some of the harder textbook problems that Ellis et al. (2019) can solve. In addition, Ellis et al. (2019) propose a method for inducing phonological representations which are universal across languages.

Beyond program synthesis, Rasin et al. (2017) use a comparable description length-based approach to unsupervised joint inference of underlying phonological forms and rewrite rule representations of phonological processes, but use a genetic algorithm to find approximate solutions. Gildea and Jurafsky (1996) and Chandlee et al. (2014) discuss supervised learning of restricted classes of finite-state transducer representations of several phonological processes (including English flapping). To date, such work either requires thousands of training observations (Gildea and Jurafsky, 1996) or has used abstracted and greatly simplified symbol inventories and training data (Chandlee et al., 2014).

Hayes and Wilson (2008), Goldsmith and Riggle (2012), and Futrell et al. (2017) propose different methods for learning probabilistic models of phonotactics, which represent gradient co-occurrence restrictions between surface segments within a word. Unlike the current implementation of SYPHON, these models include representational structures that enable them to capture certain non-local phenomena. However, because these models focus on phonotactics, they do not infer underlying forms or rules which relate underlying forms to surface forms.

Finally, much work has focused on learning representations of phonological processes as mappings that minimally violate a set of ranked or weighted constraints (Prince and Smolensky, 2004; Legendre et al., 1990), but such work has generally taken the constraint definitions as given and focused on learning rankings or weights (see *e.g.* Goldwater and Johnson, 2003; Tesar and Smolensky, 2000; Boersma and Hayes, 2001), with some exceptions (Doyle et al., 2014; Doyle and Levy, 2016).

7 Conclusion

We have presented a new approach to learning fully interpretable phonological rules from sets of related surface forms. We have shown that our approach produces rules that largely match linguists' intuition from a handful of examples and within minutes. The contributions of this paper are a novel decomposition of the global inference problem into three local problems, as well as an encoding of these problems into constraints that can be efficiently solved by an SMT solver.

References

Adam Albright and Bruce Hayes. 2002. Modeling English past tense intuitions with minimal generalization. In *Proceedings of the 2002 Workshop on Morphological Learning, Association for Computational Linguistics*.

Rajeev Alur, Rastislav Bodík, Garvit Juniwal, Milo M. K. Martin, Mukund Raghothaman, Sanjit A. Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. 2013. Syntax-guided synthesis. In *Formal Methods in Computer-Aided Design, FMCAD 2013*, pages 1–8.

Rajeev Alur, Arjun Radhakrishna, and Abhishek Udupa. 2017. Scaling enumerative program synthesis via

divide and conquer. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 319–336. Springer.

R. Harald Baayen, Richard Piepenbrock, and H van Rijn. 1993. The CELEX lexical database on CD-ROM.

Nikolaj Bjørner, Anh-Dung Phan, and Lars Fleckenstein. 2015. νz - an optimizing SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 194–199.

Paul Boersma and Bruce Hayes. 2001. Empirical Tests of the Gradual Learning Algorithm. *Linguistic Inquiry*, 32(1):45–86.

Jane Chandlee, Rémi Eyraud, and Jeffrey Heinz. 2014. Learning strictly local subsequential functions. *Transactions of the Association for Computational Linguistics*, 2:491–504.

Noam Chomsky and Morris Halle. 1968. *The Sound Pattern of English*. Studies in language. Harper & Row.

Leonardo Mendonça de Moura and Nikolaj Bjørner. 2008. Z3: an efficient SMT solver. In *TACAS*, volume 4963 of *LNCS*, pages 337–340. Springer.

Gabriel Doyle, Klinton Bicknell, and Roger Levy. 2014. Nonparametric Learning of Phonological Constraints in Optimality Theory. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, pages 1094–1103.

Gabriel Doyle and Roger Levy. 2016. Data-driven learning of symbolic constraints for a log-linear model in a phonological setting. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 2217–2226.

Greg Durrett and John DeNero. 2013. Supervised learning of complete morphological paradigms. In *Proceedings of the North American Chapter of the Association for Computational Linguistics*.

Kevin Ellis, Adam Albright, Armando Solar-Lezama, Joshua B. Tenenbaum, and Timothy J. O'Donnell. 2019. Synthesizing theories of human language with Bayesian program induction. In Prep.

Kevin Ellis, Daniel Ritchie, Armando Solar-Lezama, and Josh Tenenbaum. 2018. Learning to infer graphics programs from hand-drawn images. In *Advances in Neural Information Processing Systems 31*, pages 6059–6068. Curran Associates, Inc.

Kevin Ellis, Armando Solar-Lezama, and Josh Tenenbaum. 2015. Unsupervised learning by program synthesis. In *Advances in neural information processing systems*, pages 973–981.

Richard Futrell, Adam Albright, Peter Graff, and Timothy J. O'Donnell. 2017. A generative model of phonotactics. *Transactions of the Association for Computational Linguistics*, 5:73–86.

Daniel Gildea and Daniel Jurafsky. 1996. Learning bias and phonological-rule induction. *Computational Linguistics*, 22(4):497–530.

John Goldsmith and Jason Riggle. 2012. Information theoretic approaches to phonological structure: the case of Finnish vowel harmony. *Natural Language & Linguistic Theory*, 30(3):859–896.

Sharon Goldwater and Mark Johnson. 2003. Learning OT Constraint Rankings Using a Maximum Entropy Model. *Proceedings of the Stockholm Workshop on Variation within Optimality Theory*, pages 111–120.

Noah D Goodman, Joshua B Tenenbaum, Jacob Feldman, and Thomas L Griffiths. 2008. A rational analysis of rule-based concept learning. *Cognitive science*, 32(1):108–154.

Dan Gusfield. 1997. *Algorithms on Strings, Trees and Sequences*. Cambridge University Press.

Carlos Gussenhoven and Haike Jacobs. 2017. *Understanding Phonology*. Routledge.

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

Géraldine Legendre, Yoshiro Miyata, and Paul Smolensky. 1990. Harmonic Grammar – A formal multi-level connectionist theory of linguistic well-formedness: Theoretical foundations. Technical Report ICS # 90-5, CU-CS-465-90, University of Colorado.

David Odden. 2005. *Introducing Phonology*. Cambridge University Press.

José Oncina, Pedro García, and Enrique Vidal. 1993. Learning Subsequential Transducers for Pattern Recognition Interpretation Tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(5):448–458.

Alan Prince and Paul Smolensky. 2004. *Optimality Theory: Constraint interaction in generative grammar*. Wiley-Blackwell.

Ezer Rasin, Iddo Berger, Nur Lan, and Roni Katzir. 2017. Acquiring opaque phonological interactions using Minimum Description Length. In *Supplemental Proceedings of the 2017 Annual Meeting on Phonology*.

Iggy Roca and Wyn Johnson. 1999. *A Workbook in Phonology*. Blackwell.

Rohit Singh, Venkata Vamsikrishna Meduri, Ahmed K. Elmagarmid, Samuel Madden, Paolo Papotti, Jorge-Arnulfo Quiané-Ruiz, Armando Solar-Lezama, and Nan Tang. 2017. Synthesizing entity matching rules by examples. *PVLDB*, 11(2):189–202.

Armando Solar-Lezama. 2013. Program sketching. *International Journal on Software Tools for Technology Transfer*, 15(5-6):475–495.

Bruce Tesar and Paul Smolensky. 2000. *Learnability in Optimality Theory*. MIT Press.

Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. 2018. Programmatically interpretable reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning*, pages 5052–5061.

R. L Weide. 2014. The CMU pronouncing dictionary. Release 0.7b.

Yunhui Zheng, Vijay Ganesh, Sanu Subramanian, Omer Tripp, Murphy Berzish, Julian Dolby, and Xiangyu Zhang. 2017. Z3str2: an efficient solver for strings, regular expressions, and length constraints. *Formal Methods in System Design*, 50(2-3):249–288.