A method for projecting features from observed sets of phonological classes

Connor Mayer
connormayer@ucla.edu

Robert Daland
r.daland@gmail.com

Abstract

Given a set of phonological features, we can enumerate a set of phonological classes. Here we consider the inverse of this problem: given a set of phonological classes, can we derive a feature system? We show that this is indeed possible, using a collection of algorithms that assign features to a set of input classes and differ in terms of what types of features are permissible. This work bears on theories of both emergent and universal features, can serve as a useful component in computational models of feature learning, and provides testable predictions on the featurizations available to learners.

1 Introduction

Features are the substantive building blocks of phonological theory. They represent phonetic qualities of speech sounds, and can be used in isolation or combination to describe individual sounds or classes of sounds (e.g. Jakobson et al., 1952; Chomsky & Halle, 1968; Clements, 1985).

The goals of feature theory are to capture the following generalizations. First, segments that have common phonetic properties tend to behave alike, both within and across languages. Features allow such commonalities between sounds to be explicitly represented. For example, the English voiceless non-continuants {p, t, $\widehat{\text{tʃ}}$, k} are all produced with a complete closure of the oral cavity and no vocal fold vibration, and exactly these segments undergo the process of foot-initial aspiration. The feature notation $\begin{bmatrix} \text{-continuant} \\ \text{-voiced} \end{bmatrix}$ exposes these shared phonetic properties to the phonological grammar, and the processes which might reference them. More generally, the set of obstruents, which may be specified with the feature [-sonorant], tends to undergo similar voicing processes across languages (regressive voicing assimilation within obstruent clusters, word-final devoicing, intervocalic and/or postnasal voicing, and so on).

Second, sound changes tend to preserve phonetic qualities of affected sounds, even when the host segment is altered or destroyed. The sub-segmental representation afforded by features allows these changes to be modeled in a principled way. An instance of feature preservation was the fall of the yers in Old Church Slavonic. The front yer (a short,

unstressed, high front vowel) deleted in most prosodic positions. However, the preceding consonant typically became palatalized, thereby preserving the high and front articulations, even while the vowel segment was deleted (Carlton, 1991).

Finally, feature theory reflects featural economy and the factorial arrangement of the segmental inventory: if a language treats a particular featural contrast as distinctive, it is likely to be exploited widely throughout its inventory. In other words, segment inventories are more symmetric than might be expected if segments were the atoms of representation (Clements, 2003).

Classic texts (e.g. Chomsky & Halle, 1968) have assumed phonological features are *universal*: all the sounds in the world's languages can be described by the same finite set of features, which reflect properties of the human vocal tract and perceptual system. According to this view, speakers inherently produce and perceive speech in terms of these features because they are the substantive 'atoms' of which segments and higher prosodic constituents are composed. Children represent speech in terms of these atoms, which is why phonological processes operate on the classes they define. Feature theory is manifestly successful in explaining why many common phonological processes involve segments that share relevant phonetic properties.

However, there is evidence that many phonological processes target sets of segments that cannot be singled out by a set of phonetic properties. A canonical example is the *ruki* rule of Sanskrit, in which an underlying /s/ becomes retroflexed when it occurs later in a word than any of {r, u, k, i} (e.g. Kiparsky, 1973; Vennemann, 1974). While it has been proposed that the *ruki* process originated from the acoustic effects of these segments on neighboring sounds, e.g. a lowering of the noise frequency of a following /s/ (Longerich, 1998), no conventional feature system can pick out all four of these segments to the exclusion of others. The existence of a single, idiosyncratic rule like this is not grounds for theoretical concern. However, it has been proposed that *phonetically disparate classes* like {r, u, k, i} are much more common than would be expected under a universal feature system. Mielke (2008) conducted a survey of phonological processes in almost 600 languages. Of the classes which underwent or conditioned a phonological process, 71% could be expressed as a combination of simple features by the 'best' feature system he considered. To express the remaining 29%, additional theoretical mechanisms – such as building classes through an OR operation – would be needed. These seriously compromise the explanatory power that made feature theory appealing in the first place.

The purported ubiquity of phonetically disparate classes has generated proposals that features are *learned* and *language-specific* (e.g. Blevins, 2004; Mielke, 2008; MacWhinney & O'Grady, 2015; Archangeli & Pulleyblank, 2015): learners are able to group sounds in their languages into classes, even if they have no phonetic commonality. The striking regularities that exist across languages are explained as by-products of general human cognitive capabilities, such as categorization, sensitivity to frequency, and the ability to generalize, as well as the properties of the human vocal tract and auditory system that shape language change.

It is still unclear whether the problematic classes identified by Mielke and others can be captured by other means. It is possible that classes which superficially appear to be phonetically disparate result from interactions between phonological processes that target phonetically coherent classes. Alternatively, these classes may share phonetic similarities that have not yet been formalized in any feature system. Even if these issues are put aside, well established problems like the variable patterning of /l/ as [+continuant] or [−continuant] across languages (e.g. Kaisse, 2002; Mielke, 2008) suggest that features may be learned to some degree, or, at the very least, it is not a simple matter to enumerate the available features in advance.

The goal of this paper is not to take a strong position on the correctness of emergent feature theory, but rather to address the question of what a phonological learning system would look like under this theory. In constructing such a system, one could start with the features and derive classes, or start with the classes and derive features. Here we develop the latter approach. That is, we suppose that some mechanism has identified particular sets of segments as 'candidate' classes – which we refer to as the input. Several considerations motivate this approach: first, it is unclear how emergent features could be learned without being somehow motivated by the classes they characterize. Second, and more practically, the output of past attempts at unsupervised learning of phonological categories are classes (e.g. Goldsmith & Xanthos, 2009; Calderone, 2009; Mayer, 2018). In principle, a system that learns features from classes allows for the construction of a computational model that takes (minimally) a segmental corpus as input and outputs a featurization of the inventory. This paper describes the final stage of such a model. A related paper, Mayer (2018), details an algorithm that takes a phonological corpus and learns classes based on distributional similarity that could serve as the input to this stage, though distributional similarity is only one of the mechanisms by which classes are likely learned.

Below we illustrate how a feature system can be learned from an arbitrary input, i.e. without any reference to the phonetic properties of the segments contained in the input classes. We begin in Section 2 by formalizing our notation for feature systems. This notation and the lattice-like structures it motivates are similar to past work such as Broe (1993), although we provide a more detailed formalism, which aids in proofs of some interesting properties. Section 3 describes the *intersectional closure* of a set of classes, which is necessarily generated by any featurization of that set. Using the intersectional closure as a tool for efficient computation, Sections 4 to 7 describe a suite of algorithms for learning various types of featurizations for a set of input classes and include proofs of their soundness. Finally, in Section 8 we analyze some tradeoffs between the featurization algorithms, and discuss implications for feature theory and feature learning.

This paper makes several contributions. First, it demonstrates a method for working backwards to feature systems underpinning learned classes of sounds. Second, it provides a detailed formalization of feature systems in general. This allows careful reasoning about the expressiveness of such featurizations. Third, by comparing multiple types of algorithms, this work makes explicit predictions about what classes should be describable under each

type. Even under a universal theory of features, this will allow future research to precisely investigate the featurizations used by humans. For example, the full specification algorithm predicts that the class of non-nasal sounds should be available to speakers as a byproduct of the nasal class, which suggests that participants in an artificial grammar learning experiment should be able to effectively learn patterns involving this class. The other featurization methods to be discussed do not make this prediction. Finally, it provides the code[1] for use and extension in future research and models.

2 Definitions and notation

Let Σ denote an alphabet of segments. We will use the term *class* to mean a subset of Σ.

2.1 Classes and class systems

A *class system* $(\mathcal{C}, \Sigma)$ consists of an alphabet Σ and a set of classes $\mathcal{C}$ over that alphabet. Consider the following simple class system, which is meant to evoke a manner hierarchy.

Example: Manner hierarchy class system

- *alphabet* – {V, G, L, N, T}
- *sonorants* – {V, G, L, N}
- *non-continuants* – {N, T}
- *continuants* – {V, G, L}
- *singletons* – {V}, {G}, {L}, {N}, {T}

Fig. 1 illustrates this class system. Each node corresponds to a class. Downward arrows indicate a *parent/child* relationship. The parent/child relationship is of central importance to this work, so we formalize it carefully here.

Definition: Let $(\mathcal{C}, \Sigma)$ be a class system. $X \in \mathcal{C}$ is a *parent* of $Y \in \mathcal{C}$ (and Y is a *child* of X) if and only if

- $Y \subset X$, and
- there exists no $Z \in \mathcal{C}$ such that $Y \subset Z \subset X$

In other words, X is a parent of Y if a subset/superset relation holds, and there is no 'intervening' class between them.

[1] https://github.com/rdaland/Pheatures/

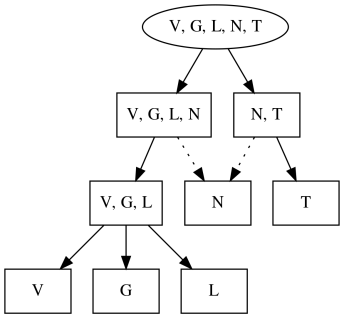

Figure 1: A manner hierarchy class system.

In Fig. 1, there is a 'path' from the alphabet through the sonorants to the continuants. This means the sonorants are a child of the alphabet, and the continuants are a child of the sonorants. The existence of this path implies the continuants are a subset of the alphabet,[2] but crucially, the continuants are not a child of the alphabet because the sonorants intervene. One reason that we depict parent/child relationships (rather than subset/superset) is to avoid crowding the graph. But there is an additional, theoretical reason – this relation is also important for the featurization algorithms we describe later. We additionally define PARENTS(Y) as the set of classes which are parents of a class Y, and CHILDREN(Y) as the set of classes which are children of a class Y.

There are some additional aspects of Fig. 1 which merit comment. First, the empty set is technically a daughter of the singletons (since it is a subset of everything) but it does not appear in the graph. This is because the empty set is a phonologically irrelevant class: it cannot partition the alphabet into segments which undergo a process and those which do not. To say that it is equivalent to the source or target of a process is equivalent to saying that the process does not happen at all.[3] For this reason, we generally omit the empty set throughout this paper. Second, the class {N} has two dotted arrows which extend to it, rather than a single solid line like the other classes. The dotted lines signify that {N} is the intersection of its parents: {V, G, L, N} ∩ {N, T} = {N}. As we will prove later, this entails that {N} can be expressed as the union of features which express {V, G, L, N} and {N, T} individually. Therefore, it does not need a new feature to distinguish it from other classes. In the next section, we develop the formal machinery to express this insight.

[2]Formally, the subset/superset relation is the *transitive closure* of the parent/child relation, and the parent/child relation is the *transitive reduction* of the subset/superset relation.

[3]Some confusion may arise with regard to SPE-style rules. In SPE, the null set symbol is used to indicate the source/target of epenthesis/deletion rules. Thus, in SPE the null set symbol is used to denote an empty string. In the present work, the null set symbol is used to denote the null set.

2.2 Feature systems and featurizations

Definition: A *feature system* is a tuple $(\mathcal{F}, \Sigma, \mathcal{V})$ where

- Σ is a segmental alphabet,
- $\mathcal{V}$ is a set of values, and
- $\mathcal{F}$ is a *featurization*: a set of features $\{f_j\}_{j=1}^{M}$, where each feature is a function $f : \Sigma \to \mathcal{V}$ mapping segments to feature values.

To illustrate, a possible feature system for the manner system of Fig. 1 is shown below in Table 1. In the next subsection we formalize featural descriptors, which relate classes and feature systems.

σ	syl	voc	apprx	son
V	+	+	+	+
G	–	+	+	+
L	–	–	+	+
N	–	–	–	+
T	–	–	–	–

Table 1: Example of a feature system.

2.3 Featural descriptors

Let $(\mathcal{F}, \Sigma, \mathcal{V})$ be a feature system. We restrict $\mathcal{V}$ to the following possibilities:

- *privative specification*: $\mathcal{V} = \{+, 0\}$
- *full specification*: $\mathcal{V} = \{+, -\}$
- *contrastive specification*: $\mathcal{V} = \{+, -, 0\}$

We will use the notation $\mathcal{V}_0$ for the set $\mathcal{V}\backslash\{0\}$, i.e. the set of non-zero values. This is because zero values are a formal mechanism to achieve underspecification, and the theoretical driver for underspecification is the idea that underspecified features are phonologically inactive (i.e. cannot define classes). Then, a *featural descriptor* $\mathbf{e}$ is a set of feature/value pairs where the values cannot be 0: i.e. $\mathbf{e} \subset \mathcal{V}_0 \times \mathcal{F}$. For example, $\mathbf{e} = \begin{bmatrix} \text{+front} \\ \text{–low} \end{bmatrix}$ is a featural descriptor. This is an *intensional* description of a class; that is, a description of a class in terms of its properties. The *extension* of a featural descriptor is the set of segments which match (at least) the feature/value pairs in the descriptor. We use angle brackets to indicate this:

$$\langle \mathbf{e} \rangle = \{x \in \Sigma \mid \forall(\alpha_k, f_k) \in \mathbf{e},\, [f_k(x) = \alpha_k]\}$$

Note that under this definition, the extension of the empty featural descriptor is Σ, since the predicate is vacuously true for all segments when $\mathbf{e}$ is empty.

We use the notation $\mathcal{V}_0^{\mathcal{F}}$ to denote the powerset of $\mathcal{V}_0 \times \mathcal{F}$, i.e. the set of all licit featural descriptors. Lastly, we define $\langle \mathcal{V}_0^{\mathcal{F}} \rangle = \{\langle \mathbf{e} \rangle \mid \mathbf{e} \in \mathcal{V}_0^{\mathcal{F}}\}$, the set of all classes described by some featural descriptor in $\mathcal{V}_0^{\mathcal{F}}$. We say that the feature system $(\mathcal{F}, \Sigma, \mathcal{V})$ generates the class system $\langle \mathcal{V}_0^{\mathcal{F}} \rangle$.

While every featural descriptor in $\mathcal{V}_0^{\mathcal{F}}$ picks out a class in $\langle \mathcal{V}_0^{\mathcal{F}} \rangle$, the two are not generally in 1-1 correspondence. This is because the same class can often be described by multiple featural descriptors. For example, under the the feature system of Table 1, the featural descriptor $\left[\begin{array}{c} +\text{voc} \end{array}\right]$ picks out the same class as $\left[\begin{array}{c} +\text{voc} \\ +\text{son} \end{array}\right]$, namely {V, G}. Moreover, the featural descriptors $\left[\begin{array}{c} +\text{syl} \\ -\text{syl} \end{array}\right]$ and $\left[\begin{array}{c} +\text{syl} \\ -\text{son} \end{array}\right]$ both pick out the empty set.

We say that a feature system $(\mathcal{F}, \Sigma, \mathcal{V})$ *covers* a class system $(\mathcal{C}, \Sigma)$ if $\mathcal{C} \subset \langle \mathcal{V}_0^{\mathcal{F}} \rangle$; in other words if the feature system provides a distinct featural representation for every class in $\mathcal{C}$. In the next subsection, we work an example to illustrate the importance of the choice of the value set in featurization.

2.4 Example: Sonorants and obstruent voicing

In this subsection we introduce a simple, 3-segment class system to illustrate the notation, as well as the difference between the privative and full specification value sets.

σ	son	vcd
R	+	+
D	0	+
T	0	0

Table 2: Sonorants and obstruents with privative specification.

Let Σ = {R, D, T}, where R is meant to evoke a sonorant, D a voiced obstruent, and T a voiceless obstruent. Accordingly we use the feature names *vcd* and *son*, but note that these descriptive labels are purely for the reader's convenience. We begin with the featurization using the private value set, shown in Table 2. The set of classes it describes, and the simplest featural descriptor for each, are shown below:

- $\langle\, [\,] \,\rangle$ = {R, D, T}

- $\langle\, [+\text{son}]\, \rangle = \{\text{R}\}$
- $\langle\, [+\text{vcd}]\, \rangle = \{\text{R, D}\}$

Note that this featurization provides (i) no featural descriptor that uniquely picks out the voiceless obstruent {T}, (ii) no way to pick out the obstruents {T} and {D} to the exclusion of {R}, (iii) no way to pick out the voiced obstruent {D} without {R}, and (iv) no way to pick out the empty set.

Next, consider the featurization in which the '0's from Table 2 are replaced with '−'s (the full specification value set):

σ	son	vcd
R	+	+
D	−	+
T	−	−

Table 3: Sonorants and obstruents with full specification.

This featurization is more expressive than the last one:

- $\langle\, [\,]\, \rangle = \{\text{R, D, T}\}$
- $\langle\, [+\text{son}]\, \rangle = \{\text{R}\}$
- $\langle\, [-\text{son}]\, \rangle = \{\text{D, T}\}$
- $\langle\, [+\text{vcd}]\, \rangle = \{\text{R, D}\}$
- $\langle\, [-\text{vcd}]\, \rangle = \{\text{T}\}$
- $\langle\, [-\text{son}, +\text{vcd}]\, \rangle = \{\text{D}\}$
- $\langle\, [+\text{son}, -\text{vcd}]\, \rangle = \varnothing$

While the privative featurization just covers three classes, the full specification featurization covers six (not counting the empty set). The ability for featural descriptors to refer to '–' values provides a greater number of ways to 'slice and dice' the alphabet. It follows that featurizations which assign more '0' values generally require more distinct feature functions to cover the same class system.

In the next section, we introduce the notion of intersectional closure, which precisely characterizes the expressiveness of any featurization. This data structure will also prove essential for efficiently assigning feature systems to a set of input classes.

3 Intersectional closure

In this section we define the *intersectional closure* of a class system $\mathcal{C}$ as the set of classes that can be generated by intersecting Σ with an arbitrary subset of classes in $\mathcal{C}$. We relate the intersectional closure to features by showing that if a feature system is expressive enough to generate all the classes in $\mathcal{C}$, it generates the intersectional closure. Then we give a dynamic programming algorithm which efficiently computes the intersectional closure.

3.1 Definitions

Definition: A collection of sets $\mathcal{C}$ is *intersectionally closed* if and only if for all $X \in \mathcal{C}$ and $Y \in \mathcal{C}$, $X \cap Y \in \mathcal{C}$.

The *intersectional closure* of a class system $(\mathcal{C}, \Sigma)$, written $\mathcal{C}_\cap$, is the smallest intersectionally closed class system which contains $\mathcal{C}$ and Σ.

Definition: $\mathcal{C}_\cap = \{ \left(\bigcap_{X_i \in P} X_i\right) \mid P \in \mathcal{P}(\mathcal{C} \cup \{\Sigma\})\}$ where $\mathcal{P}(\cdot)$ is the powerset operator.

In other words, the intersectional closure contains every class which can be generated by finite intersections of classes from $\mathcal{C}$ (and Σ), and no other classes besides these.

To illustrate this concept, we introduce the vowel inventory in Table 4. This class system will serve as a running example throughout the next four sections.

	front	mid	back
high	i y		u
mid	ɛ œ		o
low		a	

Table 4: Vowel inventory

Let $(\mathcal{C}, \Sigma)$ consist of the following classes:

- *alphabet* – {i, y, u, ɛ, œ, o, a}
- *non-low* – {i, y, u, ɛ, œ, o}
- *high* – {i, y, u}
- *front* – {i, y, ɛ, œ}
- *round* – {y, u, œ, o}
- *singletons* – {i}, {y}, {u}, {ɛ}, {œ}, {o}, {a}

$\mathcal{C}$ and $\mathcal{C}_\cap$ are depicted in Fig. 2. The difference between the two is highlighted by using red circles for the 'extra' classes.

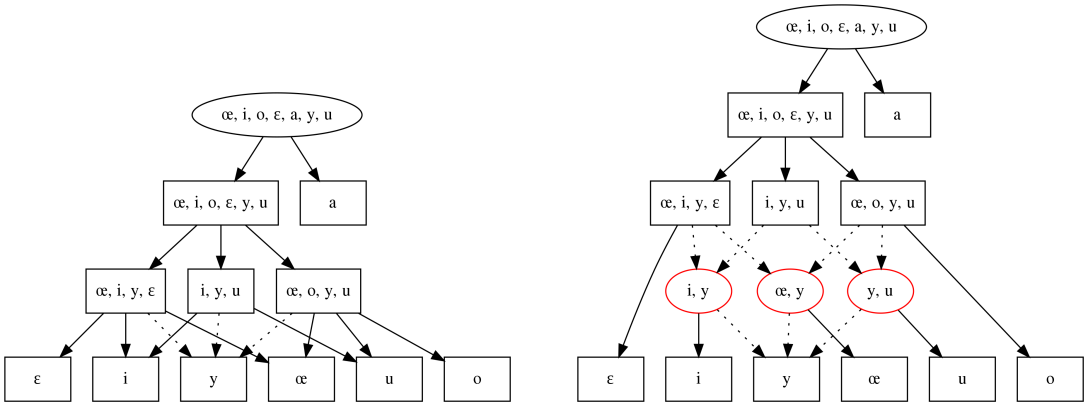

Figure 2: The original vowel inventory (left) and intersectional closure (right). Classes added in the closure are indicated with red ovals.

The key difference is that the intersectional closure contains several two-segment classes which are the intersection of larger classes. For example, the *high, front* class {i, y} is the intersection of the *high* class and the *front* class:

$$\{\text{i, y}\} = \{\text{i, y, u}\} \cap \{\text{i, y, ɛ, œ}\}$$

Note that the high, front, round class {y} has dotted lines because it is the intersection of the high/front, front/round, and high/round classes. In the next subsection, we prove that featurizations must cover intersectional closures, i.e. if a featurization is expressive enough to cover $\mathcal{C}$, it covers $\mathcal{C}_\cap$.

3.2 Feature systems generate an intersectional closure

There is a dual relationship between featural descriptors and the classes they describe: intersection of classes corresponds to union of featural descriptors. We formalize this property with the following lemma.

Featural Intersection Lemma
Let $(\mathcal{F}, \Sigma, \mathcal{V})$ be a feature system. If $\mathbf{e}_i, \mathbf{e}_j \in \mathcal{V}_0^{\mathcal{F}}$, then $\langle \mathbf{e}_i \cup \mathbf{e}_j \rangle = \langle \mathbf{e}_i \rangle \cap \langle \mathbf{e}_j \rangle$.

Proof:
The proof proceeds by showing that $\langle \mathbf{e}_i \rangle \cap \langle \mathbf{e}_j \rangle \subset \langle \mathbf{e}_i \cup \mathbf{e}_j \rangle$ and $\langle \mathbf{e}_i \cup \mathbf{e}_j \rangle \subset \langle \mathbf{e}_i \rangle \cap \langle \mathbf{e}_j \rangle$. Let $C_i = \langle \mathbf{e}_i \rangle$ and $C_j = \langle \mathbf{e}_j \rangle$. First, suppose $x \in C_i \cap C_j$. Then $x \in C_i$. By definition, x

must have the features in $\mathbf{e}_i$. Similarly, $x \in C_j$, and therefore must have the features in $\mathbf{e}_j$. Thus, x has the features in $\mathbf{e}_i \cup \mathbf{e}_j$. This shows that $C_i \cap C_j \subset \langle \mathbf{e}_i \cup \mathbf{e}_j \rangle$. Now, suppose $x \in \langle \mathbf{e}_i \cup \mathbf{e}_j \rangle$. Then x has all the features of $\mathbf{e}_i$, and so $x \in C_i$. Similarly, x has all the features of $\mathbf{e}_j$, so $x \in C_j$. Therefore $x \in C_i \cap C_j$. This shows that $\langle \mathbf{e}_i \cup \mathbf{e}_j \rangle \subset C_i \cap C_j$. Since both $C_i \cap C_j$ and $\langle \mathbf{e}_i \cup \mathbf{e}_j \rangle$ are subsets of each other, they are equal. □

We illustrate this lemma with reference to the vowel inventory system introduced above. For concreteness, let us adopt the following featurization:

σ	low	front	round	high
i	−	+	0	+
ɛ	−	+	0	−
œ	−	+	+	−
y	−	+	+	+
u	−	−	+	+
o	−	−	+	−
a	+	0	0	0

Table 5: A featurization of the vowel inventory. The low vowel is unspecified for front/round/high features; the round feature is privative.

Let $\mathbf{e}_1 = \left[\ +\text{front}\ \right]$ and $\mathbf{e}_2 = \left[\ +\text{round}\ \right]$. Then we have:

- $\langle \mathbf{e}_1 \rangle = \langle \left[\ +\text{front}\ \right] \rangle = \{\text{œ, y, ɛ, i}\}$
- $\langle \mathbf{e}_2 \rangle = \langle \left[\ +\text{round}\ \right] \rangle = \{\text{œ, o, y, u}\}$

For these values, the Featural Intersection Lemma cashes out as follows: 'the set of vowels that are both front and round' is the intersection of 'the set of vowels that are front' and 'the set of vowels that are round':

- $\langle \mathbf{e}_1 \rangle \cap \langle \mathbf{e}_2 \rangle = \langle \left[\ +\text{front}\ \right] \rangle \cap \langle \left[\ +\text{round}\ \right] \rangle = \{\text{œ, y, ɛ, i}\} \cap \{\text{œ, o, y, u}\} = \{\text{œ, y}\}$
- $\langle \mathbf{e}_1 \cup \mathbf{e}_2 \rangle = \langle \left[\ +\text{front, +round}\ \right] \rangle = \{\text{œ, y}\}$

The Featural Intersection Lemma proves that this kind of relationship holds for any pair of featural descriptors and the classes they describe.

An important consequence of this lemma is that it can be applied inductively, to relate the union of multiple featural descriptors with the intersection of multiple classes. Because the intersectional closure is defined as the intersection of arbitrarily many classes in an input $\mathcal{C}$, the Featural Intersection Lemma ends up entailing that if a featurization covers $\mathcal{C}$, it must cover the intersectional closure.

Intersectional Closure Covering Theorem

Let $(\mathcal{C}, \Sigma)$ be a class system and $(\mathcal{F}, \Sigma, \mathcal{V})$ a feature set. If $\mathcal{C} \subset \langle \mathcal{V}_0^{\mathcal{F}} \rangle$, then $\mathcal{C}_\cap \subset \langle \mathcal{V}_0^{\mathcal{F}} \rangle$.

Proof:

Let Y be an arbitrary class in $\mathcal{C}_\cap$. By definition of $\mathcal{C}_\cap$, there exist $\{X_i \in \mathcal{C}\}_{i \in I}$ (for some index set I, hereafter omitted) such that $Y = \bigcap_i X_i$. The hypothesis that $\mathcal{C} \subset \langle \mathcal{V}_0^{\mathcal{F}} \rangle$ implies that for every such X_i, there exists a featural descriptor $\mathbf{e}_i$ such that $\langle \mathbf{e}_i \rangle = X_i$. Thus, $Y = \bigcap_i X_i = X_1 \cap X_2 \cap \ldots \cap X_n$ can also be written $C = \bigcap_i \langle \mathbf{e}_i \rangle = \langle \mathbf{e}_1 \rangle \cap \langle \mathbf{e}_2 \rangle \cap \ldots \cap \langle \mathbf{e}_n \rangle$. It follows by induction using Featural Intersection Lemma that $Y = \langle \bigcup_i \mathbf{e}_i \rangle$:

$$
\begin{aligned}
Y &= \langle \mathbf{e}_1 \rangle \cap \langle \mathbf{e}_2 \rangle \cap \ldots \cap \langle \mathbf{e}_n \rangle \\
&= \langle \mathbf{e}_1 \cup \mathbf{e}_2 \rangle \cap \mathbf{e}_3 \cap \ldots \cap \langle \mathbf{e}_n \rangle \\
&= \langle \mathbf{e}_1 \cup \mathbf{e}_2 \cup \mathbf{e}_3 \rangle \cap \ldots \cap \langle \mathbf{e}_n \rangle \\
&\ldots \\
&= \langle \mathbf{e}_1 \cup \mathbf{e}_2 \cup \ldots \cup \mathbf{e}_n \rangle \\
&= \langle \textstyle\bigcup_i \mathbf{e}_i \rangle
\end{aligned}
$$

The preceding chain of logic demonstrates the following fact: if a class can be expressed as the intersection of classes in $\mathcal{C}$, then its features are the union of the features in each of those classes. The intersectional closure is defined as all possible intersections of classes in $\mathcal{C}$. Thus, if $(\mathcal{F}, \Sigma, \mathcal{V})$ covers $\mathcal{C}$, it covers the intersectional closure. □

Having illustrated the formal notation for features and the notion of intersectional closure, we turn now to a dynamic programming algorithm for efficiently calculating the intersectional closure.

3.3 An algorithm for calculating the intersectional closure

The following algorithm yields the intersectional closure of a class system $(\mathcal{C}, \Sigma)$. It bears a close resemblance to Dijkstra's shortest-paths algorithm (Dijkstra, 1959); a proof of soundness is given in the Appendix.

Ensure: $\mathcal{C}_\cap$ is the intersectional closure of the input $\mathcal{C}$

$\mathcal{C}_\cap \leftarrow \{\Sigma\}$
$\mathcal{Q} \leftarrow \mathcal{C}$

while $\mathcal{Q} \neq \varnothing$ **do**
  $X \leftarrow \textsc{dequeue}(\mathcal{Q})$
  if not $X \in \mathcal{C}_\cap$ **then**
    for $Y \in \mathcal{C}_\cap$ **do**
      $\textsc{enqueue}(\mathcal{Q}, X \cap Y)$

end for
APPEND($\mathcal{C}_\cap$, X)
end if
end while

3.4 Parenthood in the intersectional closure

As we will see shortly, the advantage of explicitly computing the intersectional closure is that *a new feature is required for all and only the classes which have a single parent in the intersectional closure*. The core reason for this is that if a class has two parents, it must be their intersection. We prove this here.

Single Parenthood Theorem
Let $(\mathcal{C}, \Sigma)$ be a class system and $Y \in \mathcal{C}_\cap$. If $X_1, X_2 \in \text{PARENTS}(Y)$, then $Y = X_1 \cap X_2$.

Proof:
First, observe that $Y \subset X_1 \cap X_2$. This follows trivially from the definition of parenthood: X_1 is a parent of Y implies $Y \subset X_1$, X_2 is a parent of Y implies $Y \subset X_2$, and so every element in Y is in both X_1 and X_2.

Now suppose that $X_1 \cap X_2 \neq Y$. The preceding logic showed that either the two are equal, or Y is a proper subset of $X_1 \cap X_2$. But the latter case creates a contradiction. By definition, $X_1 \cap X_2$ must be in the intersectional closure, and $X_1 \cap X_2 \subset X_1$ follows from fundamental properties of sets. Then $X_1 \cap X_2$ intervenes between Y and X_1, contradicting the hypothesis that Y is a daughter of X_1. Thus, $Y = X_1 \cap X_2$. □

Note that the Single Parenthood Theorem does not logically exclude the possibility that a class may have more than two parents. Rather, it guarantees that in such cases, the intersection is the same regardless of how many parents are considered. One case in which this can happen is the null set: if x, y, z are three distinct elements from Σ, then $\{x\} \cap \{y\} = \varnothing = \{y\} \cap \{z\}$. A more interesting case arose already in Fig. 2, in the intersectional closure of the vowel inventory. There, the three features *front*, *high*, and *round* give rise to three distinct 2-feature classes (featural descriptors: $\begin{bmatrix} +\text{front} \\ +\text{high} \end{bmatrix}$, $\begin{bmatrix} +\text{high} \\ +\text{round} \end{bmatrix}$, $\begin{bmatrix} +\text{round} \\ +\text{front} \end{bmatrix}$). The intersection of any pair of these is {y} (the high, front, round vowel). Thus, the set {y} has three parents, but which segments it contains is uniquely determined by any two of them.

With the necessary structures and notation in place, we now turn to the main question addressed in this paper: given a set of phonological classes, how can we generate a covering feature system? We detail four algorithms that accomplish this, differing in their assumptions about a few properties of the featurization. In the next section, we describe

the first and simplest of these: an algorithm that generates a privative feature system that covers the intersectional closure $\mathcal{C}_\cap$, given an input class system $(\mathcal{C}, \Sigma)$.

4 Privative specification

The following algorithm yields a *privative* featurization of a set of classes: that is, one where the set of legal feature values $\mathcal{V} = \{+, 0\}$. It does so by assigning a different feature/value pair, $\left[\ +f\ \right]$, to the segments in each class with a single parent.

Require: $\mathcal{C}_\cap$ is the intersectional closure of a class system $(\mathcal{C}, \Sigma)$
Ensure: $\mathcal{F}$ is a featurization over $\mathcal{V} = \{+, 0\}$ which covers $\mathcal{C}$

$\mathcal{Q} \leftarrow \mathcal{C}_\cap$
$\mathcal{F} \leftarrow \varnothing$

while $\mathcal{Q} \neq \varnothing$ **do**
  $X \leftarrow \textsc{pop}(\mathcal{Q})$
  if $|\textsc{parents}(X)| = 1$ **then**
    define $f_X : \Sigma \to \mathcal{V}$ by $f_X(\sigma) = \begin{cases} + & \text{if } \sigma \in X \\ 0 & \text{otherwise} \end{cases}$
    $\textsc{append}(\mathcal{F}, f_X)$
  end if
end while

Proof of soundness for the privative specification algorithm

A featurization algorithm is *sound* if for every class system $(\mathcal{C}, \Sigma)$, it returns a feature system which covers $\mathcal{C}$. To see that the privative specification algorithm is sound, note that every class in $\mathcal{C}_\cap$ enters the queue $\mathcal{Q}$. For an arbitrary class X in the queue, there are 3 cases. If X has 0 parents, then it is Σ, and is covered by the empty featural descriptor. If X has exactly 1 parent, then the segments in X will have the features of that parent (which uniquely pick out the parent class), plus a new feature f which distinguishes the segments in X from X's parent. If X has more than 1 parent, then Single Parenthood Theorem shows, via the Featural Intersection Lemma, that the union of features of X's parents uniquely pick out all and only the segments in X. Thus, each class which exits the queue has a set of features assigned to its segments which pick out that class uniquely. This completes the proof. □

In Fig. 3, we illustrate the outcome of applying the privative specification algorithm to (the intersectional closure of) the vowel class inventory shown in Fig. 2. The visual style is similar, but this figure contains additional annotations for the feautures themselves. The

boxes which represent the classes contain the segments in the class, followed by the list of features that are shared by all segments in the class. Recall that if a class has a feature, all descendants of the class share that feature. The 'introduction' of a feature is thus indicated explicitly by labeling and coloring the edge which points to the first/highest class whose segments share the feature. This could give the misleading impression that features are assigned to classes, so it is worth reiterating that features are maps from *segments* to values. Note that the complete featurization assigned to each segment is thus represented by inspecting the singleton sets. Finally, for readability we use familiar feature names when the feature picks out more than segment, and the segment's name when the feature only picks out a single segment. However, this is a convenience for the readers of the paper, and does not reflect the featurization algorithm. The code we supply along with this paper simply assigns an integer to each feature (F1, F2, etc.).

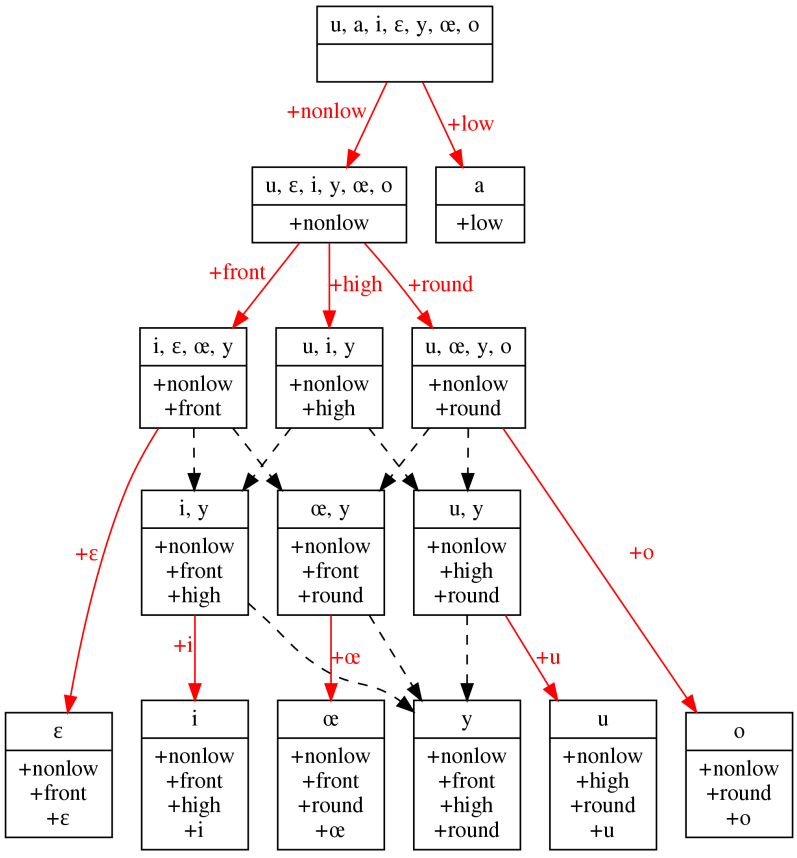

Figure 3: Yield of the privative specification algorithm.

We close this section with some observations on the properties of the privative specification algorithm and the featurization it yields.

4.1 Properties of privative specification

One point to observe is that the privative specification algorithm is *maximally conservative*. What we mean by this is that the resulting feature system generates the smallest class system that covers $\mathcal{C}$. As the Intersectional Closure Covering Theorem showed, any featurization which covers $\mathcal{C}$ will cover $\mathcal{C}_\cap$. This means that any classes which are the intersection of input classes, but which were not themselves in the input, will be 'accessible' to the output feature system. But the privative specification algorithm will not make it possible to refer to any other classes outside the intersectional closure. For example, if the input contains a $\left[\ \text{+front}\ \right]$ class and a $\left[\ \text{+round}\ \right]$ class, it must generate a $\left[\begin{array}{c}\text{+front}\\ \text{+round}\end{array}\right]$ class, but it will not 'create' a $\left[\ \text{–round}\ \right]$ class.

This might be the desired behavior. But other properties might be desired instead. For instance, one might have theoretical grounds for allowing '−' values. One might also wish to have an *efficient* feature system which minimizes the number of features needed to cover $\mathcal{C}$. It is easy to show that one can sometimes achieve a more efficient feature system by adding classes to the system. For example, the featurization shown in Fig. 3 contains ten features: *low*, *non-low*, *front*, *high*, *round*, plus five features for the individual segments that cannot be accessed as combinations of these features (note that /y/ alone can be uniquely specified using some combination of these). If the input consists of the following classes, the privative specification algorithm returns a featurization with eight features:[4]

- *alphabet* – {i, y, u, ɛ, œ, o, a}
- *non-low* – {i, y, u, ɛ, œ, o}
- *front* – {i, y, ɛ, œ}
- *back* – {u, o}
- *round* – {y, u, œ, o}
- *unround* – {i, ɛ}
- *high* – {i, y, u}
- *mid* – {ɛ, œ, o}
- *singletons* – {i}, {y}, {u}, {ɛ}, {œ}, {o}, {a}

Crucially, this featurization covers the original class system shown in Fig. 2. Thus, it uses fewer features while generating a richer class system.

[4]We will not prove this here, but you might enjoy doing this yourself using the descriptions of the intersectional closure and privative algorithm given above.

This example is presented to make two points. First, the relationship between classes in the input and the specification algorithm is not monotone. In general, adding features to a system will make more classes accessible – but in this example, a smaller number of features covers a larger class system. Thus, the minimal number of features needed to cover $\mathcal{C}$ is not predictable from a simple property, such as the total number of classes in $\mathcal{C}$. More precisely, the proof of soundness of the privative specification algorithm gives an upper bound on the features needed to cover a class system (namely, the number of classes in the intersectional closure with a single parent). We return to the issue of feature efficiency and expressiveness in Section 8. In the meantime, we turn to the second point this example makes – adding the 'right' classes to the input enables a more economical feature system. In the next sections, we explore variants of the privative specification algorithm which consider complement classes and assign '$-$' values instead of (or in addition to) '0' values.

5 Contrastive underspecification

One of the best cases for non-privative specifications arises from complement classes, such as round vs. nonround vowels, or voiced vs. voiceless obstruents. Consider a language with rounding harmony, such as Turkish. Under privative specification one would need to write one harmony rule for the [+round] feature, and an otherwise identical rule for the [+nonround] feature. By allowing features to take on opposing values, one formally recognizes the sameness of rounding with respect to the harmony process.

In canonical cases like rounding harmony and voicing assimilation, the binary feature is only relevant for certain segments. For example, in the case of rounding harmony, it is normally useful to assign the [+round] and [-round] values only to vowels. In some languages, one might wish to only assign these values to just non-low vowels, or just front vowels. In all such cases, the contrasting feature values denote complementary classes – but complements with respect to *what*?

The central insight developed in this paper is that a new feature needs to be assigned just in case a class has a single parent in the intersectional closure. This suggests that a relevant domain for complementation is with respect to the parent. This is the distinction between privative specification and contrastive underspecification: a '$-$' value is assigned when the complement of the class being processed with respect to its parent is in the input.

Require: $\mathcal{C}_\cap$ is the intersectional closure of input class system $(\mathcal{C}, \Sigma)$
Ensure: $\mathcal{F}$ is a featurization over $\mathcal{V} = \{+, -, 0\}$ which covers $\mathcal{C}$

$\mathcal{Q} \leftarrow \mathcal{C}_\cap$
$\mathcal{F} \leftarrow \varnothing$

while $\mathcal{Q} \neq \varnothing$ **do**
  $X \leftarrow$ DEQUEUE($\mathcal{Q}$)
  if $|\text{PARENTS}(X)| = 1$ **then**
    $P_X \leftarrow$ DEQUEUE(PARENTS(X))

$$\overline{X} \leftarrow \begin{cases} P_X \setminus X & \text{if } (P_X \setminus X) \in \mathcal{C} \\ \varnothing & \text{otherwise} \end{cases}$$

$$\text{define } f_X : \Sigma \to \mathcal{V} \text{ by } f_X(\sigma) = \begin{cases} + & \text{if } \sigma \in X \\ - & \text{if } \sigma \in \overline{X} \\ 0 & \text{otherwise} \end{cases}$$

    APPEND($\mathcal{F}, f_X$)
    $\mathcal{Q} \leftarrow \{x \in \mathcal{Q} | x \neq \overline{X}\}$
  end if
end while

The soundness of this algorithm follows from the soundness of the privative specification algorithm. This is because the contrastive underspecification algorithm yields a feature system which generates the same class system as privative specification does. The difference between the two is that if the input contains complement sets, then contrastive underspecification will use a single feature with '+' and '−' values, where privative specification will have two features with just '+' values.

We illustrate this algorithm using the same vowel system used in the previous section. The featurization using this algorithm is shown in Fig. 4. Note that now only nine features are required. The segment /a/, which was [+low] under the privative algorithm, has instead been featurized as [–nonlow] here. This is because the low and non-low classes are complements with respect to their parent (Σ), and both are present in the input. Contrastive [nonlow] is doing the work of privative [low] and privative [nonlow] together, so there is no need for a contrastive [low] feature.

An additional point of note is that the {y} class gets [–ɪ], [–œ], and [–u] values, and these features are now ternary. This occurs because /y/ is a complement to the clases with single parents that motivate the addition of these features.

In general, the requirements for receiving a [$-f$] value are not as strict as receiving a [+feature] value: [$-f$] classes may have more than one parent (i.e. they could be generated by intersections of other features, as {y} is here). In addition, they do not necessarily need to be siblings of the class motivating the addition of the new feature, although in Fig. 4 they happen to be. We return to this latter point in the next section. It is an empirical question whether one or both of these requirements should be in place for classes that receive [$-f$] values, but adding these restrictions introduces complications for other types of featurization presented here, such as the full specification algorithm described in Section 7.

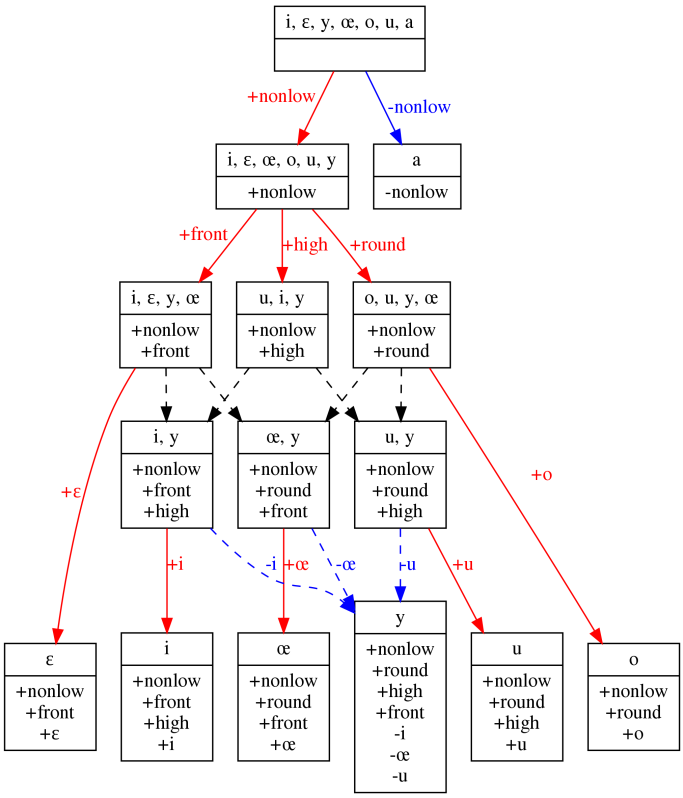

Figure 4: Yield of the contrastive underspecification algorithm.

Note that the remaining features ([front], [round], [high], [ɛ], [o]) are still privative, because their respective complements are not present in the input.

The term *contrastive underspecification* is meant to capture that features can be binary, ternary, or privative. Segments will be underspecified with respect to a feature if the relevant complement class with respect to a parent is not included in the input. If the additional complement classes described in Section 4.1 were included in the input for the vowel system, fewer features would be required. In the next section, we consider a variant of the algorithm which adds such complement classes, even if they were not present in the input. We call this variant *contrastive specification*.

6 Contrastive specification

Contrastive specification is very similar to contrastive underspecification. The key difference is that contrastive specification adds complements (with respect to the parent) to the covering. Every complement gets a '−' feature, including those which were not in the input.

This can result in adding classes that are not in the intersectional closure of the input. One way to address this is to update the intersectional closure dynamically. However, it is also possible to precompute the result (because the classes that must be added can be defined in terms of subset/superset relations, which do not depend on features); we do this as it is conceptually simpler.

We denote the function that adds complement classes with AddComplementsContrastive. When adding complement classes, the ordering in which classes are processed is crucially important. Breadth-first traversal – processing all the siblings of a class before its children – is done to avoid configurations that duplicate a feature. In addition, the order in which siblings are processed during breadth-first traversal has important consequences for the generated class and feature systems. We adopt a procedure whereby the complements of all siblings are added *simultaneously* to the class set if they are not already present. This has the potential to result in less efficient featurizations than adding the complements one-by-one as each class is processed, but it avoids imposing class hierarchies that are not motivated by the input class set. A further motivation for this scheme is that if classes are not processed simultaneously, some order must be chosen, and there is no clear motivation for choosing one over another. A detailed description of AddComplementsContrastive and considerations of other variants can be found in Appendix B.

Require: $\mathcal{C}_\cap$ is the intersectional closure of input class system $(\mathcal{C}, \Sigma)$
Ensure: $\mathcal{F}$ is a featurization over $\mathcal{V} = \{+, -, 0\}$ which covers $\mathcal{C}$

$\mathcal{Q} \leftarrow$ AddComplementsContrastive$(\mathcal{C}_\cap)$
$\mathcal{F} \leftarrow \varnothing$

while $\mathcal{Q} \neq \varnothing$ **do**
  $X \leftarrow$ dequeue$(\mathcal{Q})$
  if $|\text{parents}(X)| = 1$ **then**
    $P_X \leftarrow$ dequeue(parents(X))
    $\overline{X} \leftarrow P_X \setminus X$
    define $f_X : \Sigma \to \mathcal{V}$ by $f_X(\sigma) = \begin{cases} + & \text{if } \sigma \in X \\ - & \text{if } \sigma \in \overline{X} \\ 0 & \text{otherwise} \end{cases}$
    append$(\mathcal{F}, f_X)$
    $\mathcal{Q} \leftarrow \{x \in \mathcal{Q} | x \neq \overline{X}\}$
  end if
end while

This algorithm is sound because it considers all the classes that the privative specification

algorithm does, plus others. Thus, it necessarily covers $\mathcal{C}$.

Fig. 5 illustrates the contrastive specification algorithm using the vowel system introduced earlier. Now the complement classes with respect to their parent of the round, high, and front classes have been added, resulting in a more efficient and expressive featurization containing only binary or ternary features.

There is an important difference between this plot and previous ones. We may consider two types of plots when plotting featural relations: a *topological* plot, which plots relationships between classes using the familiar notions of the parent/child relationship developed throughout this paper, and a *featural* plot, where classes corresponding to $[\ +f\ ]$ and $[\ -f\ ]$ feature/value pairs are plotted as siblings. In all cases considered so far, these two plotting strategies have resulted in identical outcomes. With the contrastive specification algorithm on the simple vowel inventory, this is no longer the case. As discussed in the previous section, this mismatch is the result of $[\ -f\ ]$ values being assigned to classes that have multiple parents and/or are not siblings of the class motivating the new feature.

The plot in Fig. 5 is the featural plot. The topological plot is shown in Appendix B.4. The most salient difference is that the [–front] and [–round] classes are represented as siblings of the [+front] and [+round] classes in the featural plot, corresponding to the structure of the feature system. The topological plot, however, shows that these classes are not in fact siblings.

We use the featural plot here because it is more representative of the abstract structure and relationships assigned by the feature system, which subsume the topological system to some degree. Comparing the two plots provides some insight into how topological and featural relationships in a class system may diverge.

Note that the feature system yielded by contrastive specification is much more expressive than the one yielded by privative specification or contrastive underspecification. However, it is not maximally expressive, since it still contains '0' values. When a new feature is added, non-zero values are added only to classes that are descendants of the parent of the class that generates the new feature. In the class system above, /a/ is unspecified for all features except [low], since it is a descendant only of Σ.

As another example, assume a consonant inventory where stridents are daughters of coronals, and coronals are daughters of Σ. Then contrastive specification will create a [–coronal] class (all noncoronals) and a [–strident] class. The latter class will include all coronal nonstridents, but it will not include labials or other noncoronal nonstridents. Thus, while the *coronal* feature assigns a '+' or '−' value to every segment, the *strident* feature assigns a '0' value to noncoronals. If it is desired to eliminate all '0' values, one can do complementation with respect to Σ rather than the single parent. That is the final variant we discuss – full specification.

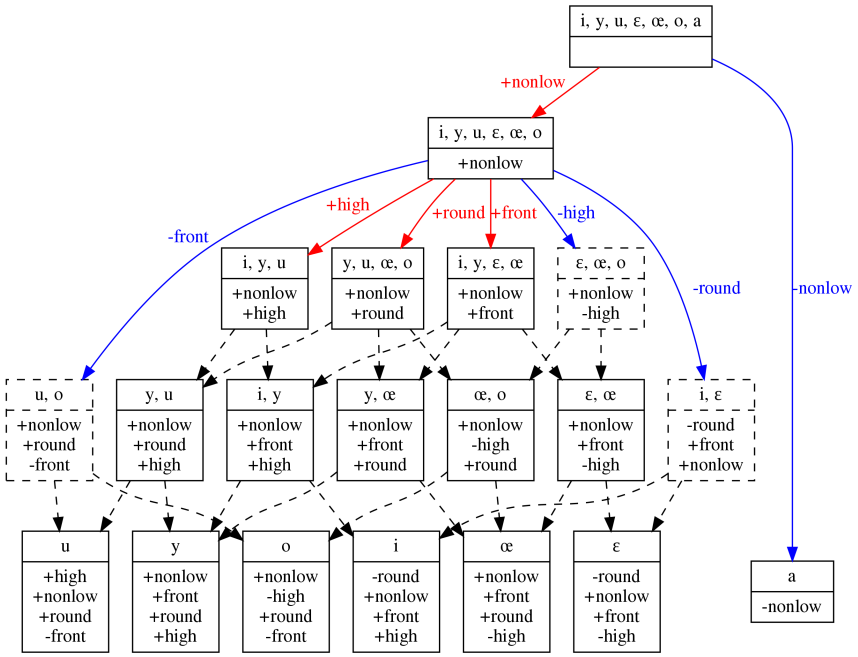

Figure 5: Class system and featurization yielded by contrastive specification. Classes explicitly added by the algorithm are indicated with dashed boxes. Classes added due to the addition of these classes to the intersectional closure are not highlighted.

7 Full specification

Full specification differs from contrastive specification in that complementation is calculated with respect to the whole alphabet, rather than the parent class. Therefore, it is algorithmically almost the same as contrastive specification. As with contrastive specification, the complement classes are precomputed and added to the intersectional closure in breadth-first search order, and siblings are processed simultaneously. We denote this process as AddComplementsFull: see Appendix B for a detailed description.

Require: $\mathcal{C}_\cap$ is the intersectional closure of input class system $(\mathcal{C}, \Sigma)$
Ensure: $\mathcal{F}$ is a featurization over $\mathcal{V} = \{+, -\}$ which covers $\mathcal{C}$

$\mathcal{Q} \leftarrow$ AddComplementsFull$(\mathcal{C}_\cap)$
$\mathcal{F} \leftarrow \varnothing$

while $\mathcal{Q} \neq \varnothing$ **do**
  $X \leftarrow$ dequeue$(\mathcal{Q})$

if $|\textsc{parents}(X)| = 1$ **then**
$\overline{X} \leftarrow \Sigma \setminus X$
define $f_X : \Sigma \to \mathcal{V}$ by $f_X(\sigma) = \begin{cases} + & \text{if } \sigma \in X \\ - & \text{otherwise} \end{cases}$
$\textsc{append}(\mathcal{F}, f_X)$
$\mathcal{Q} \leftarrow \{x \in \mathcal{Q} | x \neq \overline{X}\}$
end if
end while

The full specification algorithm is sound for the same reason that the contrastive specification algorithm is – it considers a superset of classes that the privative specification algorithm does, and thus it covers the input.

The featural plot is shown in Fig. 6, which uses the same vowel system from previous sections. The topological plot is shown in Appendix B.4. The most salient difference, in addition to those present in the case of contrastive specification, is that /a/ is a child of Σ in the featural plot. This reflects the point at which it is necessary to create a [nonlow] feature to distinguish the [+nonlow] segments from the full set of segments. The topological plot shows that /a/ is not a child of Σ, and actually has multiple parents.

The number of features in the full featurization is the same as the contrastive featurization, but now /a/ is fully specified for all features, and several new classes have been introduced as a consequence, significantly altering the overall structure of the class system.

The key way in which full specification differs from contrastive specification is that no privative specification can occur whatsoever. For example, if a single feature [+nasal] is used to pick out nasal segments, then the feature system will also generate the class [–nasal] consisting of all non-nasal segments. According to our understanding of nasal typology, this is probably not the desired behavior for the nasal feature (e.g. Trigo, 1993). However, it is possible to avoid generating a [–nasal] class by ensuring that the nasals are generated as the union of pre-existing features, rather than needing their own feature. For example, if [–continuant] picks out the nasals and oral stops, while [+sonorant] picks out vowels, glides, liquids, and nasals, then the nasal class is picked out by $\begin{bmatrix} \text{–continuant} \\ \text{+sonorant} \end{bmatrix}$. Therefore, the set of all non-nasals will not generated as a complement class because the [+nasal] feature is not generated at all. A desirable property of this solution is that the following classes fall out: continuant non-sonorants (fricatives), continuant sonorants (approximants), and non-continuant non-sonorants (stops and affricates). Less desirably, this solution fails to transparently represent nasal spreading processes; for example, vowel nasalization cannot be described as continuancy or sonorancy assimilation. Thus, the cross-linguistic behavior and learnability of classes like [–nasal] has the potential to inform feature theory. We take up this and other issues in Section 8.

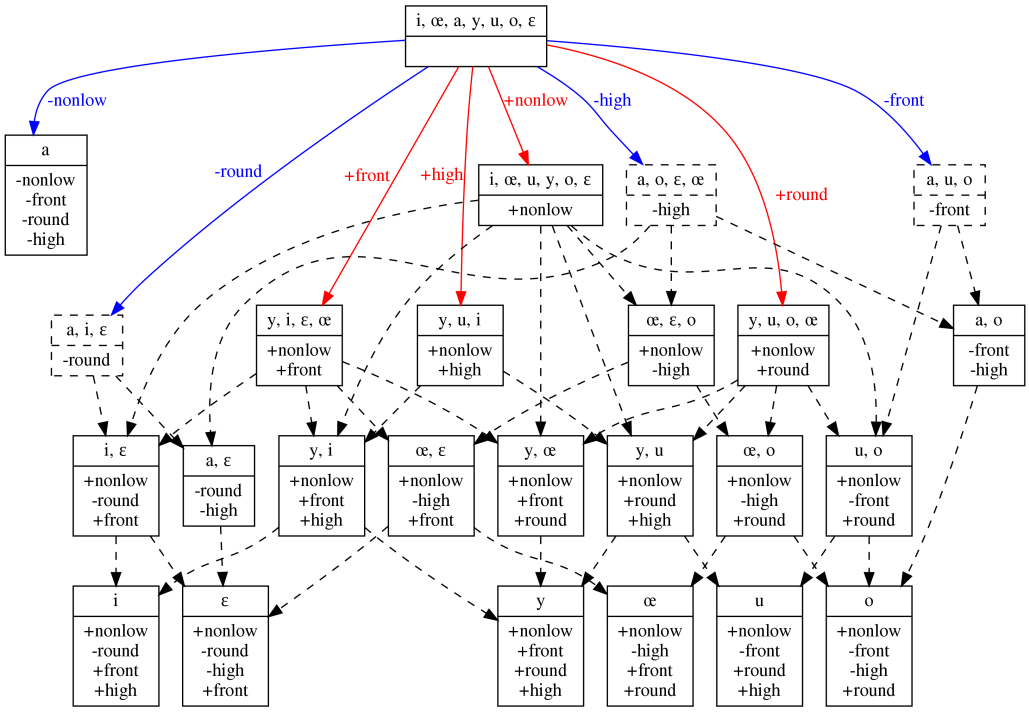

Figure 6: Class system and featurization yielded by full specification. Classes explicitly added by the algorithm are indicated with dashed boxes. Classes added due to the addition of these classes to the intersectional closure are not highlighted.

8 Discussion

In this paper, we have described a number of algorithms which assign a featurization to a set of classes, such that every class in the input can be picked out by a featural description. We gave several variants of the algorithm, differing in how conservative they are with respect to the input. The most conservative algorithm assigns a privative specification, i.e. feature functions which only pick out positively specified elements. Contrastive underspecification is achieved with the same algorithm, except that a negative specification is assigned just in case the complement of a class with respect to its parent class is in the input. Contrastive specification is similar, except that a negative specification is assigned even if the complement with respect to the parent was not in the input. Full specification is similar to contrastive specification, except the complement is taken with respect to the entire segmental alphabet. In this section, we discuss some outstanding issues, such as feature economy, how the current work bears on feature theory, and applications toward a richer theory of feature learning.

8.1 Feature efficiency and expressiveness

Here we present some examples which illustrate a little more about the expressiveness of class systems.

Let $\mathcal{C} = \{\{\sigma\} \mid \sigma \in \Sigma\}$; that is, the input consists of all and only the singleton sets. For convenience, we will refer to this as the *singleton input*. Privative specification will yield a featurization with n features, where n is the cardinality of Σ. This is because each segment gets its own feature, since the only parent of each segment is Σ. This featurization will generate only the classes in the input (and Σ, and $\varnothing$).

The opposite extreme is obtained by the *singleton complement* input – where the input consists not of all singleton sets, but the complement of each singleton set: $\mathcal{C} = \{\Sigma \backslash \{\sigma\} \mid \sigma \in \Sigma\}$. It is possible to show that when the privative specification algorithm is given this input, it generates the full powerset of Σ – every possible subset gets a unique combination of features. This follows from the fact that any set can be defined by listing the features for the segments not contained in it. Thus, privative specification is still compatible with a maximally expressive system.

The powerset of Σ is also generated by running the full specification algorithm on the singleton input. Thus, there are cases where a more conservative algorithm yields the same class system as a less conservative algorithm (albeit with a different number of features). In fact, it is generally true that the more conservative algorithms can achieve the same level of expressiveness as any less conservative algorithm, by virtue of including the relevant complement classes in the input. For example, if all complement classes with respect to Σ are included, the privative specification algorithm yields the same class system as the full specification one does, although with twice the number of features (the singleton complement input discussed above is a special case of this). Moreover, contrastive underspecification, contrastive specification, and full specification all yield the same featurization (as well as the same class system) if every relevant complement class is included. In short, the algorithms can yield radically different class systems depending on their input – but all can be made highly expressive by tailoring the input appropriately.

8.2 Relation to feature theory

As the examples in the preceding section illustrate, the most conservative algorithms (privative specification and contrastive underspecification) are able to yield class systems that are as expressive as the less conservative algorithms. However, the converse is not true. For example, full specification cannot yield a class system as unexpressive as the singleton input does under privative specification. We regard this kind of question as an interesting area for future work, but, as working phonologists, we also want to know: which is the best algorithm to use? Put another way, what matters for feature systems? One principle is that a feature system is good to the extent that learned features render the grammar simpler and/or more insightful. For example, the use of '+' and '−' values yields insight if

both values behave in the same way with respect to a harmony or assimilation process.

The received wisdom of the field generally recognizes the following cases:

- treat certain features as binary: e.g. all segments are either [+son] or [-son]
- treat certain features as privative: e.g. nasals are [+nasal] and all others are [0nasal]
- treat most features as ternary: e.g. all obstruents are [+voiced] or [-voiced], but sonorants are simply [0voiced]

Out of the algorithms we have discussed here, only the contrastive algorithms are capable of yielding a featurization which creates all three feature types. The distinction between contrastive underspecification and contrastive featurizations depends on whether complements of input classes with respect to their parents must also be in the input (which perhaps corresponds to phonological activeness) or can be defined implicitly. This is an issue that can be resolved empirically.

Here are the conditions under which the constrastive underspecification algorithm creates those three types of feature functions:

- binary features are generated when a class X and its complement $\Sigma \setminus X$ are both in the input
- privative features are generated when a class X is in the input, but no complement (with respect to any ancestor, including its parent, Σ, and any intervening classes) is
- ternary features are generated when a class X is in the input, and its complement $\overline{X}$ with respect to its parent other than Σ is in the input

For reasons of space, we do not prove that those are the correct conditions. Instead, we present an example which generates privative, binary, and ternary features. Let $\mathcal{C}$ include the following:

- *inventory* – {a, i, u, l, r, m, n, ŋ, p, t, k, b, d, g}
- *consonants* – {l, r, m, n, ŋ, p, t, k, b, d, g}
- *sonorants* – {a, i, u, l, r, m, n, ŋ}
- *obstruents* – {p, t, k, b, d, g}
- *coronal* – {n, l, r, t, d}
- *vowels* – {a, i, u}
- *nasals* – {m, n, ŋ}

- *voiceless* – {p, t, k}
- *voiced* – {b, d, g}
- *labial* – {m, p, b}
- *dorsal* – {ŋ, k, g}
- *liquids* – {l, r}
- *lateral* – {l}
- *rhotic* – {r}

The class system that results from running the contrastive underspecification algorithm on this input is shown in Fig. 7. The features [cons] and [son] are binary because each one partitions Σ. The features [LAB], [COR], [DOR], [nas] and [liquid] are privative, because their complement (with respect to every ancestor) is not included in the input. The remaining features [vcd] and [lat] are ternary, because their complements (with respect to the parent, which is not Σ) are included in the input.

The reader may be interested in investigating what happens to the 'voicing' feature if the input includes the class of all phonetically voiced segments (i.e. $\Sigma \setminus \{p, t, k\}$).

It is our hope that the algorithms described in this paper might be used in generating explicitly testable empirical hypotheses on learning phonological features. Varying the input classes and the featurization method generates different predictions about the available phonological classes in a language. This is particularly true in the cases of the contrastive and full specification algorithms, where new classes are inferred based on the relationships between classes in the input. These featurizations provide a starting point for hypotheses that are testable in phonological experiments. For example, are speakers able to infer the existence of productive phonological classes for which the only evidence in the input is that the complement (with respect to some ancestor) behaves productively?

We also note briefly that this work bears on underspecification theory (Archangeli, 1988). The appropriate application of underspecification has been somewhat controversial in the past (e.g. Steriade, 1995). A benefit of the algorithms presented here is that they provide a completely deterministic method for generating underspecification, depending only on the input classes and the featurization method used. This is perhaps similar to hierarchical decision-tree systems (e.g. Dresher, 2003), except that in such models, the hierarchical ordering of features must be specified by the analyst, while here it falls out naturally from the relations between the input classes. An unambiguous method for determining underspecification is doubtless of value to the field, and we leave as a question for future research how closely the methods described here line up with past analyses, and whether the predictions they make are borne out empirically.

Something that has not been considered in this paper is the possibility of applying − feature values to complements with respect to an ancestor other than the parent or Σ. This

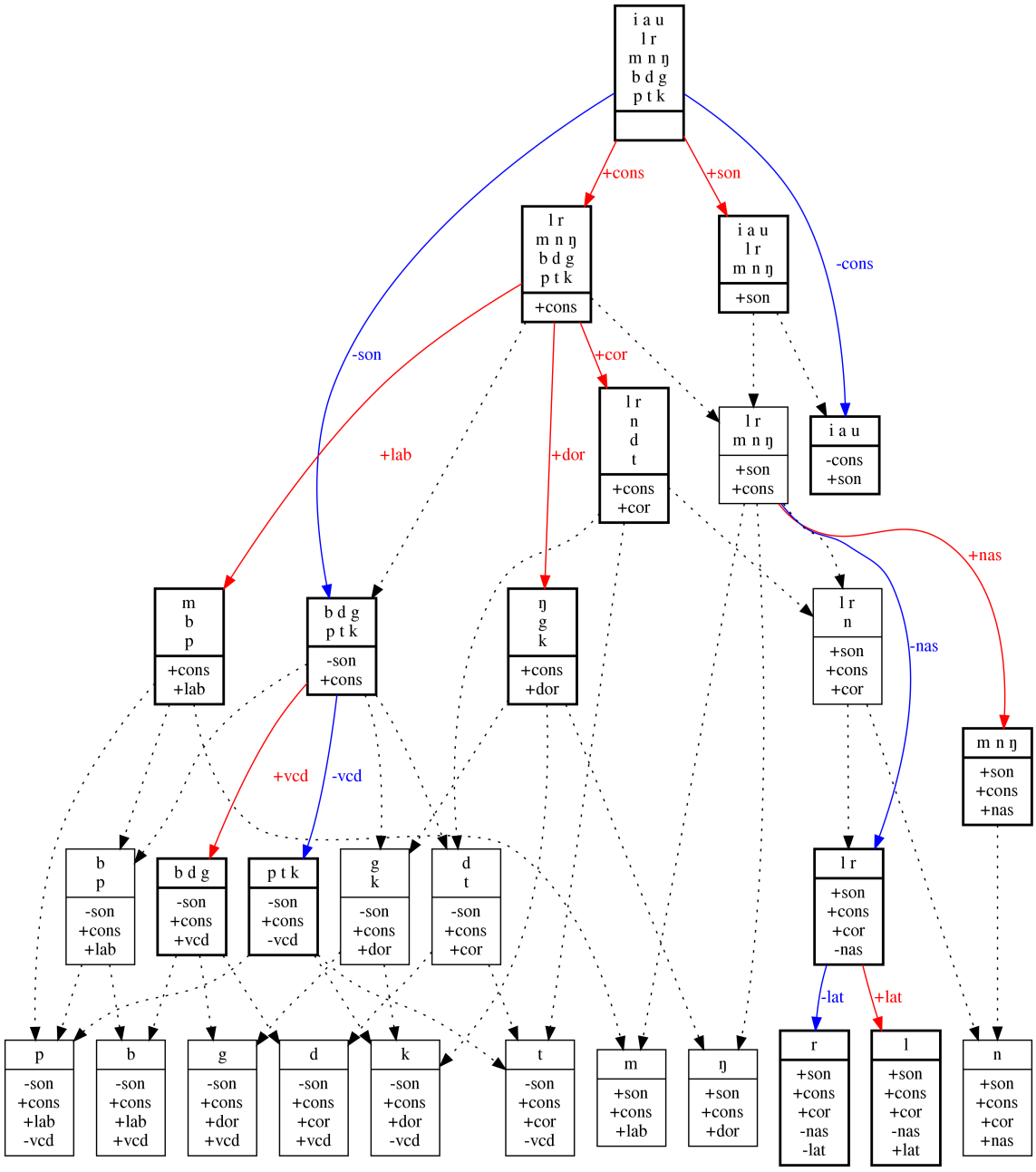

Figure 7: The output of contrastive underspecification on a large class system.

bears directly on where we want underspecification to occur. For example, we may want to specify every coronal obstruent as either [+strident] or [-strident], and all non-

coronals as [0strident]. It is less clear, though, how we should specify coronal sonorants. We do not put forth a concrete proposal for whether and how this should be done, but leave it as a possible area for future research informed by empirical phonological evidence.

8.3 Feature learning

An additional consideration associated with feature theory is that the features be learnable. We are not aware of any proposals to date which have been fleshed out enough to be carefully tested, and which have proven empirically adequate. For example, Lin (2005) showed that unsupervised clustering on acoustic data was able to distinguish manner of articulation well, but not place. Conversely, unsupervised clustering on articulatory data is better able to distinguish place features, but poor at manner (Mielke, 2012). Such phonetically-based methods are able (in principle) to identify features corresponding to acoustic, articulatory, or perceptual properties, but these provide limited insights into the phonetically disparate classes described in the introduction, and into which classes are phonologically active in a particular language.

Conversely, while distributional approaches (e.g. Goldsmith & Xanthos, 2009; Calderone, 2009; Mayer, 2018) have the potential (in principle) to identify both phonetically coherent and phonetically disparate classes to the extent that they are reflected in their distribution, they are blind to the phonetic properties that inform speakers' intuitions about similarities between sounds, and suffer from the presence of distributional noise.

It seems likely to us that progress will come from integrating multiple sources of information, with phonetic information providing a outline of possible classes, and distributional information shedding additional light on how and whether these classes (and possibly others) are used in a language. We see potential in methods like those described in a related paper by Mayer (2018), which uses a combination of vector embedding (representing sounds numerically as points in space based on their distributional properties), Principal Component Analysis, and clustering algorithms to explicitly extract classes from a corpus. Incorporating phonetic information with distributional information may improve the performance of such models. Alternatively, a Bayesian approach, where phonetic similarity serves as an initial prior on segmental classes and considerations of their distribution inform the likelihood function, may also hold promise. A challenge for the extraction of classes from phonetic data is that classes of sounds are almost always similar only on a subset of phonetic dimensions (e.g. sonorants are articulatorily heterogeneous, but have similar acoustic properties), and the use of dimensionality reduction techniques such as Principal Component Analysis is likely to be useful in teasing apart these sources of coherence.

In any case, what must be learned under models of feature learning, even those that do not incorporate distributional learning, are *classes*, with features subsequently derived from their relation. We see the question of how phonological classes are learned as one of great interest, and we hope that the work here serves as a useful component in learning models and a tool in future empirical investigations.

9 Conclusion

This paper provides a detailed formalization of the properties of phonological feature systems and describes algorithms for efficiently calculating various types of featurizations of a set of input classes. An implementation of these algorithms is available for use in further research. This work provides a stronger formal grounding for the study of phonological features, may serve as a useful component in computational models of feature learning, and the predictions made by the algorithms for various inputs and featurization types provide useful, testable empirical hypotheses for future experimental phonological research.

A Soundness proof: Intersectional closure algorithm

The proof goes by induction. First, we show that every class which can be generated by the intersection of 0 classes (Σ) or 1 class from $\mathcal{C}$ (i.e. $\mathcal{C}$ itself) belongs to $\mathcal{C}_\cap$. Next, we prove the induction step: if every class that can be generated by the intersection of n classes from $\mathcal{C}$ is in $\mathcal{C}_\cap$, then every class that can be generated by the intersection of $n+1$ classes from $\mathcal{C}$ is in $\mathcal{C}_\cap$.

Observe that $\mathcal{C}_\cap$ is initialized to contain Σ. Moreover, $\mathcal{Q}$ is initialized to contain every class in $\mathcal{C}$. Each of these must be 'transferred' to the intersectional closure because they do not belong to it already (dequeued from $\mathcal{Q}$, and appended to $\mathcal{C}_\cap$). This demonstrates that every intersection of 0 classes (Σ) and 1 class from $\mathcal{C}$ (namely, $\mathcal{C}$ itself) belongs to $\mathcal{C}_\cap$.

Now, suppose that the algorithm has guaranteed that every intersection of n classes from $\mathcal{C}$ is in $\mathcal{C}_\cap$. If there exists a $Y \in \mathcal{C}_\cap$ which can be written as the intersection of $n+1$ classes, i.e. $Y = X_1 \cap X_2 \cap \ldots \cap X_{n+1} = Y' \cap X_{n+1}$ where $Y' = X_1 \cap X_2 \cap \ldots \cap X_n$. Since every intersection of n classes is in $\mathcal{C}_\cap$, Y' must be in $\mathcal{C}_\cap$. Now, regardless of whether X_{n+1} was transferred from $\mathcal{Q}$ to $\mathcal{C}_\cap$ before or after Y' was, there was some point at which one was in $\mathcal{Q}$ and the other in $\mathcal{C}_\cap$. When the **for** loop dequeued the one in $\mathcal{Q}$, it added the intersection of this one with all others in $\mathcal{C}_\cap$ – i.e. $Y' \cap X_{n+1}$. Either this class was already in $\mathcal{C}_\cap$, or else it was not; and in the latter case, it was transferred. Thus, all sets generated by the intersection of $n+1$ classes from $\mathcal{C}$ are in $\mathcal{C}_\cap$. This completes the proof.

B The breadth-first algorithm for adding complement classes

The contrastive and full featurization algorithms add classes to $\mathcal{C}_\cap$ during their execution. In this section, we provide descriptions of the AddComplementsContrastive and AddComplementsFull algorithms introduced in Sections 6 and 7. We then motivate the use of breadth-first traversal using examples where traversing the classes in an arbitrary order produces spurious features, and discuss considerations on the order in which siblings are processed.

B.1 The algorithms

The algorithms for adding complement classes traverse $\mathcal{C}_\cap$ and, for classes with a single parent, add their complement with respect to their parent (contrastive specification) or Σ (full specification) to the class system. In order to avoid specifying spurious features, the order in which classes are processed is crucial. Specifically, $\mathcal{C}_\cap$ must be traversed in breadth-first order: that is, processing all the siblings of a class before processing any of its children. We provide some examples where this results in more efficient feature systems in Appendix B.2. We conjecture that breadth-first traversal will always produce identical or smaller feature systems than traversal in an arbitrary order, but do not provide formal proofs here.

In addition, siblings are processed simultaneously, and all their generated complements (if any) are added to the class system simultaneously. The motivation for this is discussed in Appendix B.3.

Below is the algorithm for AddComplementsContrastive:

Require: $\mathcal{C}_\cap$ is the intersectional closure of input class system $(\mathcal{C}, \Sigma)$

$\mathcal{Q} \leftarrow \{\Sigma\}$

while $\mathcal{Q} \neq \varnothing$ **do**
  $p \leftarrow$ dequeue($\mathcal{Q}$)
  Children $\leftarrow$ children(p)
  ChildComplements $\leftarrow \varnothing$
  while Children $\neq \varnothing$ **do**
    $c \leftarrow$ dequeue(Children)
    if $|$parents(c)$| = 1$ **then**
      $\overline{c} \leftarrow p \setminus c$
      append(ChildComplements, $\overline{c}$)
    end if
  end while
  $\mathcal{C}_\cap \leftarrow$ IntersectionalClosure($\mathcal{C}_\cap$, $\mathcal{Q}'$ = ChildComplements)
  NewChildren $\leftarrow$ children(p)
  append($\mathcal{Q}$, NewChildren)
end while

The algorithm for AddComplementsFull is identical, except the complement is taken with respect to Σ rather than the parent (i.e. the line $\overline{c} \leftarrow p \setminus c$ is replaced with $\overline{c} \leftarrow \Sigma \setminus c$).

B.2 Breadth-first vs. arbitrary traversal

Recall that a new feature only needs to be added when a class has a single parent. The contrastive and full specification algorithms add the complement with respect to the parent and the alphabet, respectively. These new classes alter the class structure, meaning that a class that has a single parent at one point may have two parents after a new class is added. Thus redundant classes (and hence features) may be added if a class with a single parent is processed before another class whose complement would become a parent of the first class.

Consider the input class structure shown in Fig. 8, and suppose we are processing it using full specification (i.e. adding complement classes with respect to Σ).

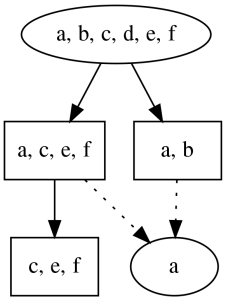

Figure 8: A simple class system.

If the class {c, e, f} is processed before {a, b}, its complement with respect to Σ, {a, b, d} will be added to the class system. When {a, b} is processed later, its complement with respect to Σ, {c, e, d, f} is added to the class system, and becomes an additional parent to {c, e, f}. This results in the feature system shown on the left side of Fig. 9.

Note that the only purpose of F3 is to differentiate the newly added class {a, b, d}, whose presence is ultimately unmotivated since {c, e, f}, the class which generated it, ends up having two parents.

Now consider the same input, but suppose that we process {a, b} before {c, e, f} (i.e. in breadth-first order). Processing {a, b} adds its complement with respect to Σ, {c, d, e, f}, which becomes the second parent to {c, e, f}. Now when {c, e, f} is processed, its complement with respect to Σ is not added because it does not have only a single parent. This results in the feature system shown on the right side of Fig. 9.

Note that the breadth-first feature system is exactly as expressive as the arbitrary system, with the exception of the unmotivated class {a, b, d}. Both cover the original input. A similar example can be generated for the contrastive case.

Thus using breadth-first traversal produces a smaller featurization system that differs only in its ability to generate unmotivated classes. We conjecture that using breadth-first traversal guarantees that when a class is processed, all of its parents that will be added to the input by the end of the algorithm will have already been added, but we leave the proof as a question for future research.

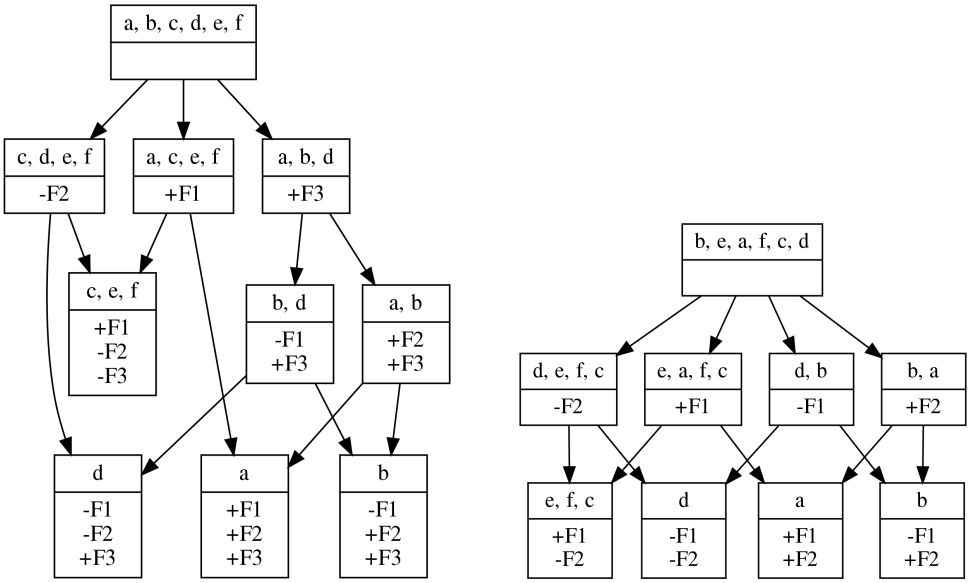

Figure 9: The classes generated after running AddComplementsFull if classes are processed in an arbitrary order (left) and in breadth-first order (right).

B.3 Considerations on the ordering of siblings

Although breadth-first traversal gives us a rough guide for how to process classes, it does not completely determine the order. The question of the order in which siblings should be processed is still unanswered. Here, too, ordering proves to be important for the resulting class system. Consider the input shown in Fig. 10, and suppose this time that we are running the contrastive specification algorithm.

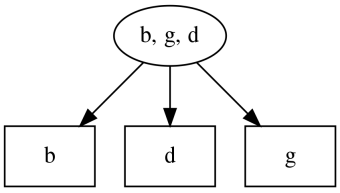

Figure 10: A simple class system.

Suppose we process the class {b} before either of the other classes. This will result in the complement of {b} with respect to Σ, {d, g}, being added to the class system. This is shown on the left side of Fig. 11.

This is troubling, however, because it predicts that the class {d, g} should be available in the phonology, while the similar classes {b, g} and {b, d} should not. This prediction

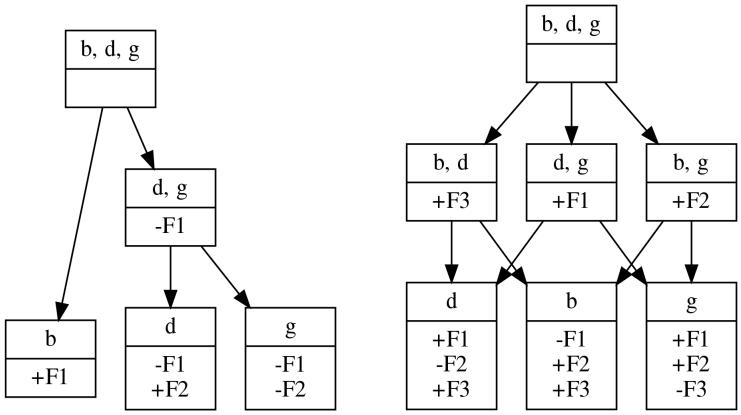

Figure 11: The resulting feature systems when siblings are processed sequentially (left) and simultaneously (right) using the contrastive specification algorithm.

is unmotivated by the class structure, and occurs in some form regardless of which class is processed first.

In light of this observation, and given the lack of an obvious principled way to choose which class should be processed first, we process siblings *simultaneously*: that is, the complements of *all* siblings are calculated, and added to the class system at the same time. In this case, the resulting class system is shown on the right side of Fig. 11.

This feature system is less efficient, in the sense that it requires more features, but the overall structure is the one best motivated by the input classes. If the simpler structure is indeed the desired one, the class {d, g} can simply be added to the input.

When the full specification algorithm is run on the input in Fig. 10 with sequential processing, a similarly arbitrary class structure is generated, although in this case it involves two of the three possible two-segment subclasses rather than only one.

B.4 Topological plots of the contrastive and full specification outputs

The plots below show the topological plots of the output of the contrastive and full specification algorithms on the vowel inventory used throughout the paper. Classes explicitly added by the algorithm are indicated with dashed lines. Classes added due to the addition of these classes to the intersectional closure are not highlighted. The parent/child relationship is maintained in the graph, but featural siblings (i.e. +/- pairs) are not necessarily plotted at the same level.

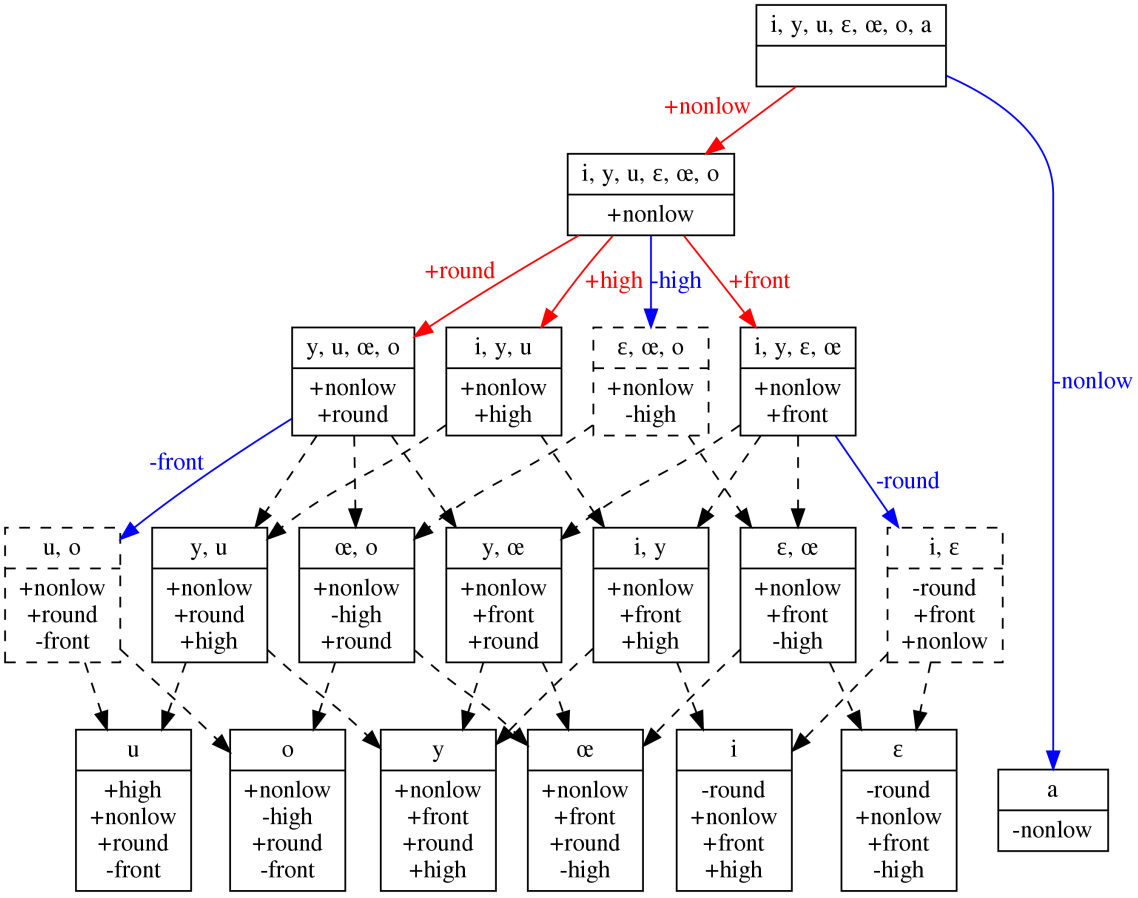

Figure 12: The topological plot of the output of the contrastive specification algorithm.

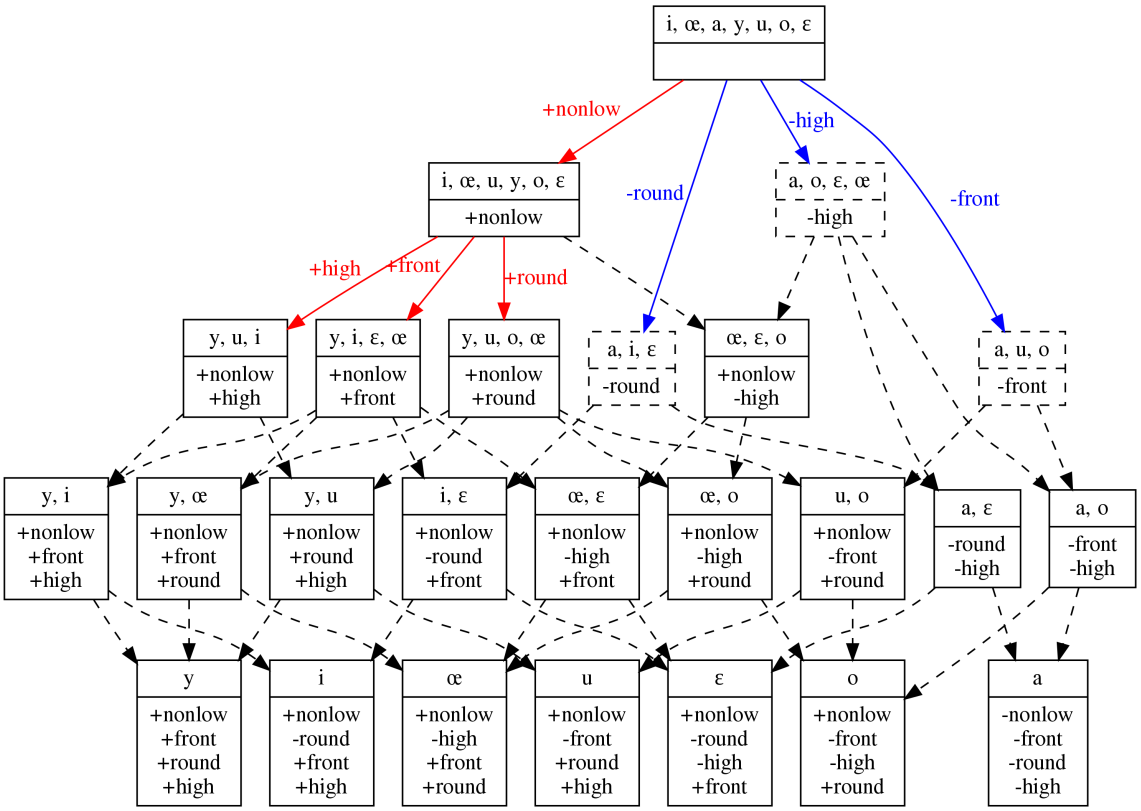

Figure 13: The topological plot of the output of the full specification algorithm.

References

Archangeli, D. (1988). Aspects of underspecification theory. *Phonology*, *5*, 183-207.

Archangeli, D., & Pulleyblank, D. (2015). Phonology without universal grammar. *Frontiers in Psychology*, *6*, 1229.

Blevins, J. (2004). *Evolutionary phonology: The emergence of sound patterns.* Cambridge: Cambridge University Press.

Broe, M. (1993). *Specification theory: The treatment of redundancy in generative phonology* (Unpublished doctoral dissertation). University of Edinburgh.

Calderone, B. (2009). Learning phonological categories by independent component analysis. *Journal of Quantitative Linguistics*, *16*.

Carlton, T. R. (1991). *Introduction to the phonological history of the Slavic languages.* Bloomington, Indiana: Slavica Publishers.

Chomsky, N., & Halle, M. (1968). *The sound pattern of English.* New York: Harper & Row.

Clements, G. N. (1985). The geometry of phonological features. *Phonology Yearbook*, *2*, 225-252.

Clements, G. N. (2003). Feature economy in sound systems. *Phonology*, *20*, 287-333.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, *1*, 269-271.

Dresher, E. (2003). The contrastive hierarchy in phonology. In D. C. Hall (Ed.), *Toronto working papers in linguistics 20: Special issue on contrast in phonology.* University of Toronto.

Goldsmith, J., & Xanthos, A. (2009). Learning phonological categories. *Language*, *85*, 4-38.

Jakobson, R., Gunnar, C., Fant, M., & Halle, M. (1952). *Preliminaries to speech analysis: The distinctive features and their correlates.* Cambridge, MA: MIT Press.

Kaisse, E. M. (2002). *Laterals are [-continuant].* MS, University of Washington.

Kiparsky, P. (1973). Phonological representations. In O. Fujimura (Ed.), *Three dimensions of linguistic theory* (p. 1-136). Tokyo: TEC Co.

Lin, Y. (2005). *Learning features and segments from waveforms: A statistical model of early phonological acquisiton* (Unpublished doctoral dissertation). UCLA.

Longerich, L. (1998). *Acoustic conditioning for the RUKI rule.*

MacWhinney, B., & O'Grady, W. (Eds.). (2015). *The handbook of language emergence.* Chichester: John Wiley & Sons.

Mayer, C. (2018). *An algorithm for learning phonological classes from distributional similarity* (Unpublished master's thesis). University of California, Los Angeles.

Mielke, J. (2008). *The emergence of distinctive features.* Oxford: Oxford University Press.

Mielke, J. (2012). A phonetically-based metric of sound similarity. *Lingua*, *1222*, 145-163.

Steriade, D. (1995). Markedness and underspecification. In J. Goldsmith (Ed.), *The handbook of phonological theory* (p. 114-175). Oxford/Cambridge, MA: Blackwell.

Trigo, R. L. (1993). The inherent structure of nasal segments. In M. Huffman & R. Krakow (Eds.), *Nasality.* San Diego: Academic Press.

Vennemann, T. (1974). Sanskrit *ruki* and the concept of a natural class. *Linguistics*, *130*, 91-97.