A method for projecting features from observed sets of phonological classes

Mayer, Connor
connormayer@ucla.edu

Daland, Robert
r.daland@gmail.com

Abstract

This paper describes a collection of dynamic programming algorithms which assign features to a set of phonological classes. The input consists of a set of classes, each containing one or more segments. If a class can be generated as the union of existing features (i.e. as the intersection of already-processed classes), those features are propagated to every segment in the class. A new feature/value pair must be assigned otherwise, and we show that this occurs just in case the class has a single parent in the intersectional closure of the input. The algorithm comes in four flavors, which differ with respect to complementation and how negative values are assigned. We show that these variants yield *privative specification*, *contrastive underspecification*, *contrastive specification*, and *full specification*, respectively. This algorithm sheds light on how a feature system may be derived from a set of learned classes, bearing on theories of emergence in phonology and providing testable predictions on the kinds of featurization available to language 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. 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. 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 (e.g. Clements, 2003).

For example, the English voiceless stops {p, t, $\widehat{\text{t}\int}$, 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). 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).

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, it is also clear 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 led some researchers to propose that distinctive 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. Under this emergent view of features, 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.

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 $\left[\ +\text{continuant}\ \right]$ or $\left[\ -\text{continuant}\ \right]$ across languages suggest that features may be learned to some degree (e.g. Kaisse, 2002; Mielke, 2008).

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 need to 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 phonological *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 segmental inventory.

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 segment. We begin by formalizing our notation for feature systems. We then describe the *intersectional closure* of a set of classes, which must be generated by any featurization of that set. Using the intersectional closure as a tool for efficient computation, we describe a suite of algorithms for learning various types of featurizations for a set of input classes and prove their soundness. Finally, 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 the code[1] for use in future research. Third, it provides a detailed formalization of several featurization algorithms. This allows careful reasoning about the expressiveness of such featurizations. Finally, by comparing multiple types of algorithms, this work makes explicit

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

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.

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. Over the course of this paper, we will introduce three different class systems, which will serve as recurring examples. Here is the first. The class and segment labels are 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 $Y \subset Z \subset X$

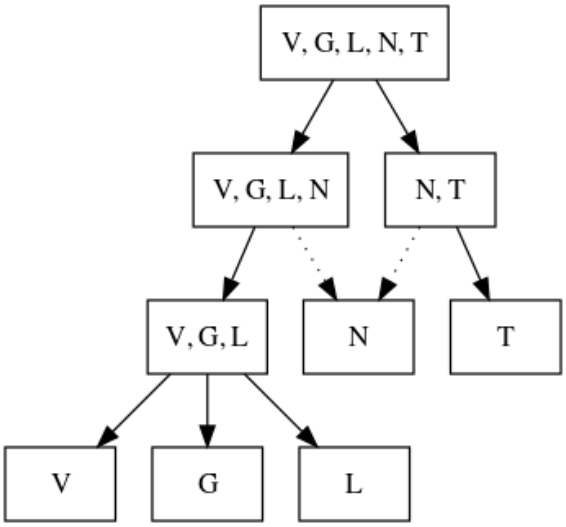

Figure 1: A manner hierarchy class system (recurring example).

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

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). It is convenient to depict parent/child relationships (rather than subset/superset) to avoid crowding the graph. But 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 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; and 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] 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

[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.

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. We will also prove that the converse is true as well: a class needs a new feature just in case it cannot be expressed as the union of the features of its parents. This turns out to mean that a class needs a new feature if and only if it has a single parent. However, this is not guaranteed to be true for the input class system $\mathcal{C}$; instead, it holds for a derived class system $\mathcal{C}_\cap$ we call the intersectional closure. Some careful formalization is necessary to express these insights.

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 $(\mathcal{V} \setminus \{0\})$, 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 $\langle \mathcal{V}_0^{\mathcal{F}} \rangle$.

Note that 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 $\begin{bmatrix} \text{+voc} \end{bmatrix}$ picks out the same class as the featural descriptor $\begin{bmatrix} \text{+voc} \\ \text{+son} \end{bmatrix}$, namely {V, G}. Moreover, the featural descriptors $\begin{bmatrix} \text{+syl} \\ \text{–syl} \end{bmatrix}$ and $\begin{bmatrix} \text{+syl} \\ \text{–son} \end{bmatrix}$ 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.

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.

σ	son	vcd
R	+	+
D	0	+
T	0	0

Table 2: Sonorants and obstruents with privative specification.

The set of classes it describes, and the simplest featural descriptor for each, are shown below:

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

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. This data structure will prove essential for efficiently assigning feature systems.

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 last of our three running examples, the vowel inventory in Table 4.

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

$\mathcal{C}$ is depicted in Fig. 2; the intersectional closure $\mathcal{C}_\cap$ is depicted in Fig. 3. The difference between the two is highlighted by using red boxes for the 'extra' classes.

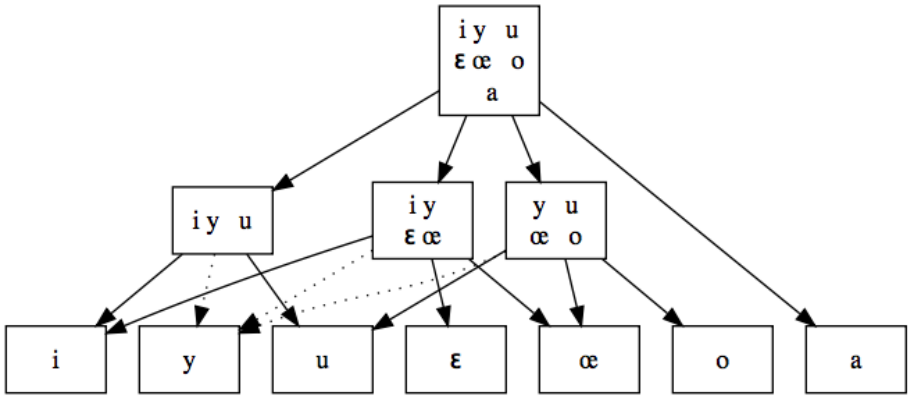

Figure 2: A vowel inventory.

The key difference is that the intersectional closure contains several 2-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.

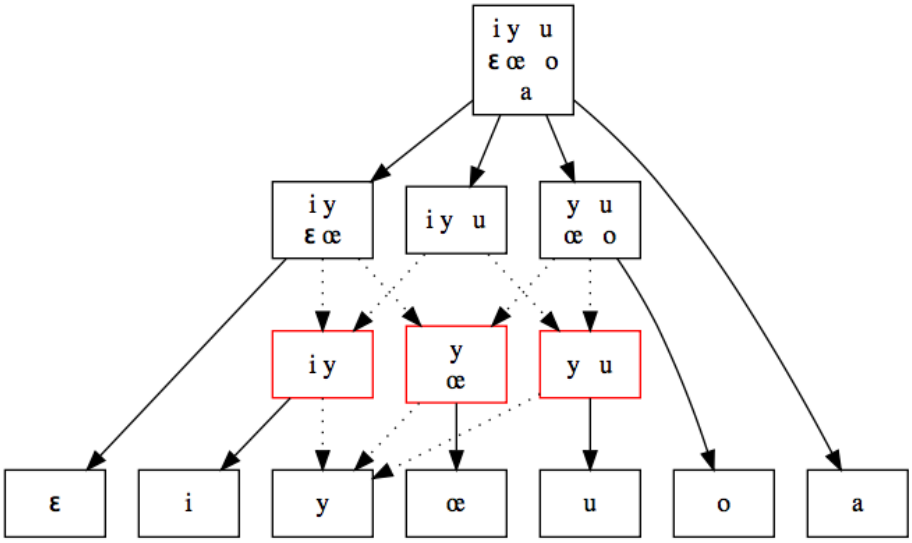

Figure 3: Intersectional closure of the vowel inventory.

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:

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':

σ	nonlow	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.

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

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**
  $X \leftarrow$ DEQUEUE($\mathcal{Q}$)
  if not $X \in \mathcal{C}_\cap$ **then**
    for $Y \in \mathcal{C}_\cap$ **do**
      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 $X_1, X_2 \in$ 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. 3, 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 3 parents, but which segments it contains is uniquely determined by any pair of them.

In the next section, we give an algorithm which generates a privative feature system that covers the intersectional closure $\mathcal{C}_\cap$, given the input of a class system $(\mathcal{C}, \Sigma)$.

4 Privative specification

The following algorithm yields a privative specification by assigning a different feature $\begin{bmatrix} +\text{f} \end{bmatrix}$ 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 get 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. 4, we illustrate the outcome of applying the privative specification algorithm to (the intersectional closure of) the vowel class inventory shown in Fig. 3. We employ several conventions to jointly optimize readability and informativity. First, classes which have a single parent are visually highlighted using a thick, red border. These represent the classes that cannot be featurized simply by the union of their parents' features. Second, the arrow which leads to each such class is annotated with the feature/value pair that is used to distinguish it. This represents the 'point' at which each feature is assigned. This could give the misleading impression that features are assigned to classes; so it is worth reinforcing here that features are maps from *segments* to values. As a consequence, when the algorithm assigns a feature/value pairs to every segment in a class, the feature/value pair is automatically inherited by every descendant of the class. The set of features that is shared by all members of a class is represented under the node. The complete featurization assigned to each segment is thus represented by inspecting the singleton sets in the bottom row. For readability, we use feature names that are familiar from phonological theory when the feature picks out more than segment in isolation (but note that the algorithm knows nothing of phonetic substance – as far as it is concerned, they are just arbitrary symbols). When the feature only picks out one segment, we name the feature after the segment.

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

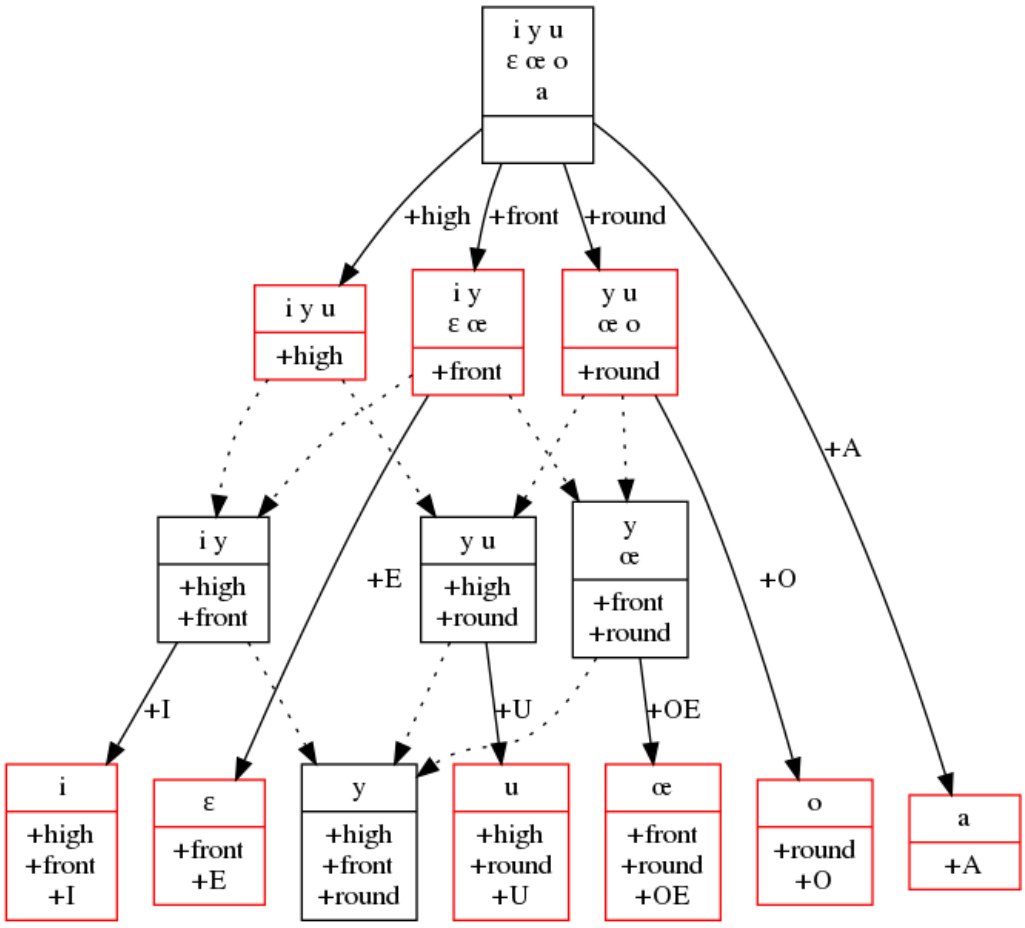

Figure 4: Yield of the privative specification algorithm.

possible to refer to any other classes outside the intersectional closure. For example, if the input contains a [+front] class and a [+round] class, it must generate a $\begin{bmatrix} +\text{front} \\ +\text{round} \end{bmatrix}$ class, but it will not 'create' a [–round] class.

This might be the desired behavior. But other properties might be desired instead. For instance, one might have theoretical grounds for wishing to allow '−' values. One might also wish to have an *efficient* feature system – one 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. 4 contains nine features: *front*, *high*, *round*, plus six features for the individual segments that cannot be accessed as combinations of front, high, and round (note that /y/ alone can be uniquely specified using these three features). If the input consists of the following classes,

the privative specification algorithm returns a featurization with seven features:[4]

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

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

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 $\mathcal{C}$. To be more precise, 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 the Discussion section. In the meantime, we turn to the second point this example makes – adding the 'right' classes to the input is what enabled 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.

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

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. 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**
  $X \leftarrow \textsc{dequeue}(\mathcal{Q})$
  if $|\textsc{parents}(X)| = 1$ **then**
    $P_X \leftarrow \textsc{dequeue}(\textsc{parents}(X))$

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

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

    $\textsc{append}(\mathcal{F}, f_X)$
    $\mathcal{Q} \leftarrow \{x \in \mathcal{Q} | x \neq \overline{X}\}$
  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 on the three-segment system {R, T, D} discussed before. Suppose that the input consists of the following:

- *obstruents* – {D, T}
- *sonorants* – {R}
- *voiced obstruents* – {D}
- *voiceless obstruents* – {T}

Then contrastive underspecification will yield the following featurization (or one which is equivalent to it, but with inverse signs):

σ	obstr	vcd
R	–	0
D	+	+
T	+	–

Table 6: Sonorants and obstruents with contrastive underspecification.

The term *contrastive underspecification* is meant to capture that features can be binary or privative. Segments will be underspecified with respect to a feature if the relevant complement class is not included in the input. For example, in Table 6 the segment R is not specified for voicing – but it would have been if the input had included the class {R, D}. In the next section, we consider a variant of the algorithm which adds the complement class, even if it was 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 search – processing all the siblings of a class before its children – is done to avoid configurations that duplicate

a feature. A detailed description of AddComplementsContrastive and proof of its correctness can be found in the Appendix.

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. Note that the feature system yielded by contrastive specification is much more expressive than the one yielded by privative specification. However, it is still not maximally expressive, since it still contains '0' values. When a new feature is added, nonzero values are added only to classes that are descendants of the parent of the class that generates the new feature. For example, suppose that 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.

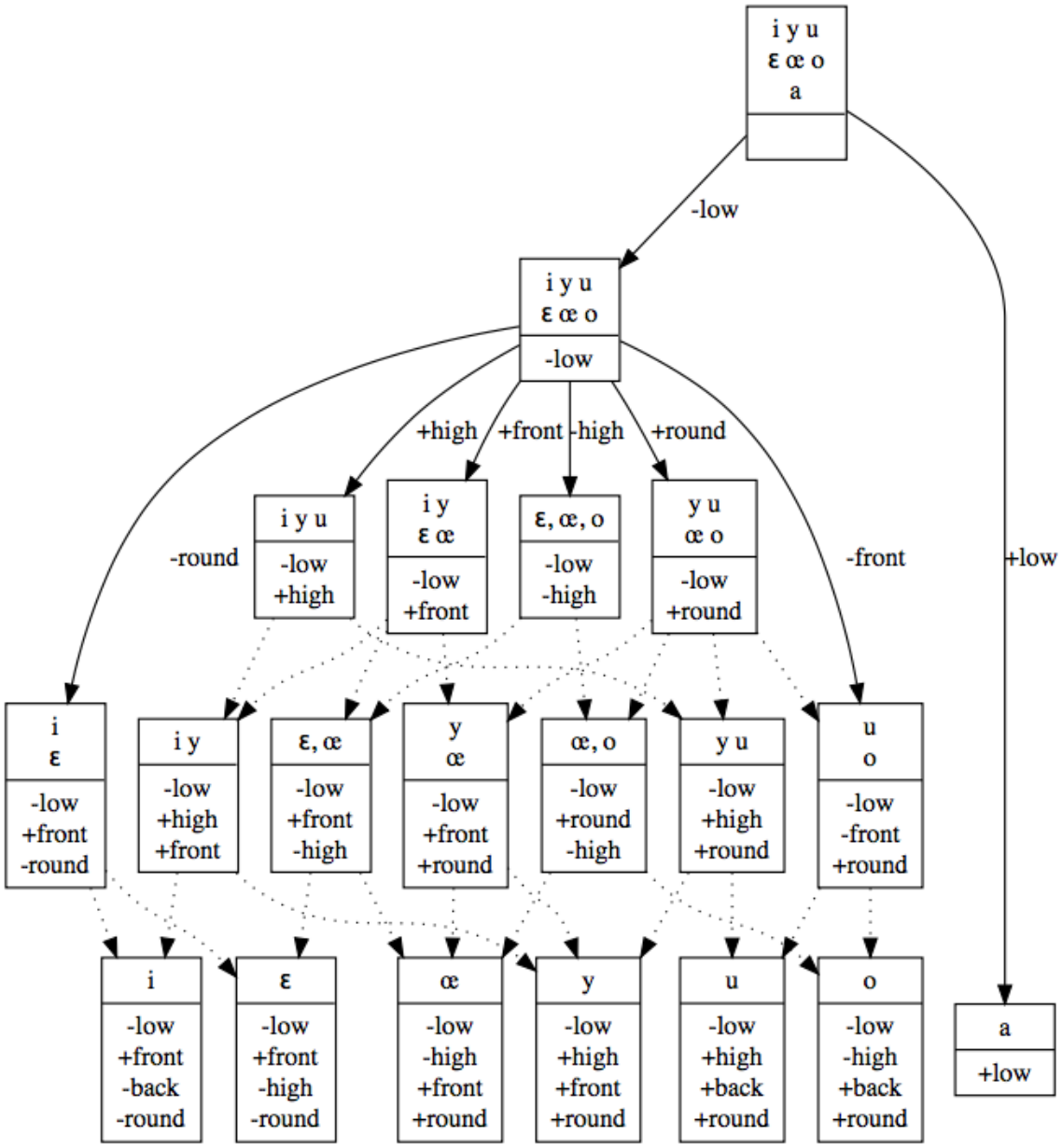

Figure 5: Class system and featurization yielded by contrastive specification.

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. We denote this process as AddComplementsFull: see the Appendix for a detailed description and proof of correctness.

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 $|\text{parents}(X)| = 1$ **then**
    $\overline{X} \leftarrow \Sigma \setminus X$
    define $f_X : \Sigma \to \mathcal{V}$ by $f_X(\sigma) = \begin{cases} + & \text{if } \sigma \in X \\ - & \text{otherwise} \end{cases}$
    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 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 the Discussion.

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*. Consider what happens when the privative specification algorithm is run on the singleton input. It will yield a featurization with n features, where n is the cardinality of Σ. This is because each segment gets its own feature. 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 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. 6. 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?

8.3 Feature learning

An additional consideration associated with feature theory is that the features be learnable. There exist various proposals as to how features might be learned, e.g. from acoustic data (?, ?), from articulatory data (?, ?), or from distributional statistics (?, ?). To date, every proposal that has been fleshed out enough to be tested has proven inadequate. It seems likely to us that progress will come from integrating multiple sources of information. In this section, we sketch an approach being undertaken in ? (?) detailing how the algorithms described in this paper might be integrated with the identification of classes using distributional and phonetic criteria to learn a feature system.

A Bayesian approach to the learning of phonological classes seems promising for several reasons. Broadly speaking, this consists of some objective function that we try to maximize by considering various configurations of input classes.

First, a central observation in phonological theory is that similar sounds seem to behave in similar ways across languages, whether in phonotactic constraints or phonological alternations. Approaches that take feature systems to be learned and language-specific account for this by positing groupings based on phonetic similarity between sounds (be this articulatory, acoustic, or perceptual) and the propensities of such groups to undergo similar types of sound change (e.g. Blevins, 2004; Mielke, 2008). Such measures of phonetic similarity serve as a useful prior in a Bayesian system, stipulating that phonological classes should tend to be phonetically cohesive along one or more dimensions, and suggesting an initial classification based on the phonetic properties of the segments in a particular language. It is worth noting that a multidimensional approach is crucial here: for example, although the sonorant/obstruent distinction is widespread in languages, sonorants are articulatorily heterogenous, with little in common except voicing. Quantificational measures that distinguish between obstruents and sonorants have been proposed, however, based on acoustic and aerodynamic measurements (?, ?) and on laryngeal measurements (Mielke, 2012). Conversely, classes based on place of articulation are (not surprisingly) more apparent from articulatory data than from any other source (Mielke, 2012).

With these priors in place, candidate classes can be determined from both phonetic data, as described above, and from distributional data, where we predict that segments

forming a class will tend to occur in the same types of environments and undergo the same alternations (see e.g. Goldsmith & Xanthos, 2009; Peperkamp et al., 2006). This distributional information supports which of the possible phonetically homogenous classes are active in a language, and crucially, whether there are active phonologically disparate classes.

The learning process consists of adding classes and accepting them or rejecting them depending on whether their resulting featurization improves the objective function. The likelihood of a data set given a proposed featurization can be calculated using something like the Hayes and Wilson (2008) constraint learner, which given a data set and a featurization of its segments, produces a set of learned constraints and a likelihood measure. An additional benefit of a Bayesian approach is that it constraints the number of proposed classes. It would be trivial to provide a class system that is an excellent fit to the data by simply featurizing the power set of the alphabet, but it is simple to configure the objective function to penalize such excess.

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. We believe that this work provides a stronger formal grounding for the study of phonological features, and that 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 order to avoid specifying spurious features, the order in which classes are considered is crucial: specifically, $\mathcal{C}_\cap$ must be traversed in breadth-first order, and the children of each node must be traversed from largest to smallest. This section provides an example of why this is the case, describes the algorithm, and provides a proof of its correctness. Contrastive specification will be used as an example, followed by a generalization to full specification.

Consider the intersectional closure shown in Fig. **??**. Suppose we process the class {ɔ} before either of the classes {oe, y, ɛ, i} or {u, y, i, ʊ}. This would add a new complement class {oe, u, y} since {ɔ} has a single parent. However, as can be seen in Fig. 5, this complement is actually unnecessary because once the complements of the two larger classes specified above are calculated, {ɔ} will have two parents and can be picked out by the union of their features rather than requiring a new feature. This new feature effectively duplicates the work that existing features can already do.

Notice that this is only important for classes that have a single parent in the intersectional closure of the input, because the algorithm is only sensitive to the distinction between one and more than one parent. If a class with more than one parent has an additional parent added after it is processed, there is no change to the resulting poset or featurization aside from having a new feature/value pair associated with the segments of that class.

By processing classes that are higher in the poset (in the sense of having fewer edges between them and Σ) before processing lower ones, we can guarantee that when any class with a single parent in the input is processed, all of its parents that will be added by the contrastive algorithm will already be in the poset, and hence no unnecessary complements/features will be added. The algorithm that does this is a modified version of the standard breadth-first search (BFS) algorithm, which traverses a graph by considering all the sisters of any node before moving to its children. This algorithm differs from standard BFS algorithms because it modifies the graph as it moves through it: when a class's children are to be added to the queue, the algorithm looks at whether it is the only parent of each child. If so, it adds the complement of that child with respect to itself as one of its children. It then looks at whether its other children are subsets of this new class, and, if so, removes them from the list of children to be added.

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

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

while $\mathcal{Q} \neq \varnothing$ **do**
  $X \leftarrow$ DEQUEUE$(\mathcal{Q})$
  $C \leftarrow$ SORTLARGETOSMALL(CHILDREN(X))
  $C' \leftarrow \varnothing$
  while $C \neq \varnothing$ **do**
    $c \leftarrow$ DEQUEUE(C)
    if $|$PARENTS$(c)| = 1 \wedge c \notin D$ **then**
      $\overline{c} \leftarrow X \setminus c$
      $\mathcal{C}_\cap \leftarrow$ INTERSECTIONALCLOSURE$(\mathcal{C}_\cap, \mathcal{Q}' = \{\overline{c}\})$
      for $c' \in C$ **do**
        if $c' \subset \overline{c}$ **then**
          REMOVE(C, c')
        end if
        APPEND$(C', \overline{c})$
        APPEND$(D, \overline{c})$
      end for
      APPEND(C', c)
      APPEND(D, c)
    end if
  end while
  ENQUEUE$(\mathcal{Q}, C')$
end while

The algorithm above guarantees that at the time the number of parents of c is checked, all of the parents that would be added by the algorithm will already be present if c only had a single parent in the input.

Proof: Consider two classes X and Y, each with a single parent, and assume without loss of generality that Y is processed after X. Suppose that when Y is processed, $\overline{Y}$, the complement of Y with respect to its parent Z that is added at this point, becomes a parent of X.

If $\overline{Y}$ is a parent of X, then X must either be a daughter or a descendant of Z. If X is a descendant of Z but not a daughter, then by definition of the algorithm X cannot have already been processed: the node currently being processed is Y, Y is a daughter of Z, and all daughters of Z are processed before any further descendants of Z. This results in

a contradiction, and thus X must be a daughter of Z.

$X \cap Y = \varnothing$, or $\overline{Y}$ would not be a parent of X. If X has already been processed, it must be the case that $\overline{X} = Z \setminus X$ has already been added to the poset since Z is the only parent of X. Because $X \cap Y = \varnothing$, it must be the case that $Y \subset \overline{X}$. But then $Y \subset \overline{X} \subset Z$, which means Y cannot be a daughter of Z. This results in a contradiction and completes the proof.

TODO: Talk about processing children in sorted order and give example of where this makes a difference.

The algorithm for adding complement classes in full specification is virtually identical to the one presented above, except that we take the complement with respect to the alphabet and not the parent class (i.e., $\overline{c} \leftarrow \Sigma \setminus c$).

TODO: The proof for the full version is actually a little tricky, and doesn't really follow nicely from the proof of the contrastive version above. There might be something linking the two I'm missing. It may also be the case that this isn't true!

What we want to prove is that if a class X with a single parent has been processed already (and hence a new complement $\overline{X}$ wrt Σ added), no later class Y will add a new complement that becomes an additional parent of X.

Some observations, assuming X has already been processed and $\overline{Y}$ ends up being a parent of X as above. Assume that X_p is the single parent of X prior to this and Y_p is the single parent of Y:

- $X \cap Y = \varnothing$ or $\overline{Y}$ wouldn't be a parent of X.
- $\overline{X}$ will have already been added and it MUST be the case that $Y_p \subseteq \overline{X}$. If this weren't the case, both $\overline{X}$ and Y_p would be parents of Y, and $\overline{Y}$ wouldn't be added. This doesn't preclude $\overline{X}$ from being the parent of Y.
- It follows that $X \cap Y_p = \varnothing$, or $\overline{X}$ wouldn't be a superset of Y_p.
- It must be the case that $X_p \cap \overline{Y} = X$, by the definition of parenthood earlier in the paper.
- By the same token, $X_p \not\subset \overline{Y}$ and $\overline{Y} \not\subset X_p$, or the number of parents X has wouldn't change.
- I think it follows from this that $X_p \cap Y \neq \varnothing$, or $\overline{Y}$ would be a superset/subset of X_p.

I've been trying to produce a proof by contradiction using these facts, but nothing has been forthcoming. It's possible that this is just not true, but I'm also having a hard time creating a counter-example. Leaving this on the back burner for now.

References

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.

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.

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

Hayes, B., & Wilson, C. (2008). A maximum entropy model of phonotactics and phonotactic learning. *Linguistic Inquiry*, *39*(3), 379 - 440.

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.

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.

Peperkamp, S., Calvez, R. L., Nadal, J., & Dupoux, E. (2006). The acquisition of allophonic rules: Statistical learning with linguistic constraints. *Cognition*, *101*, B31-B41.

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.

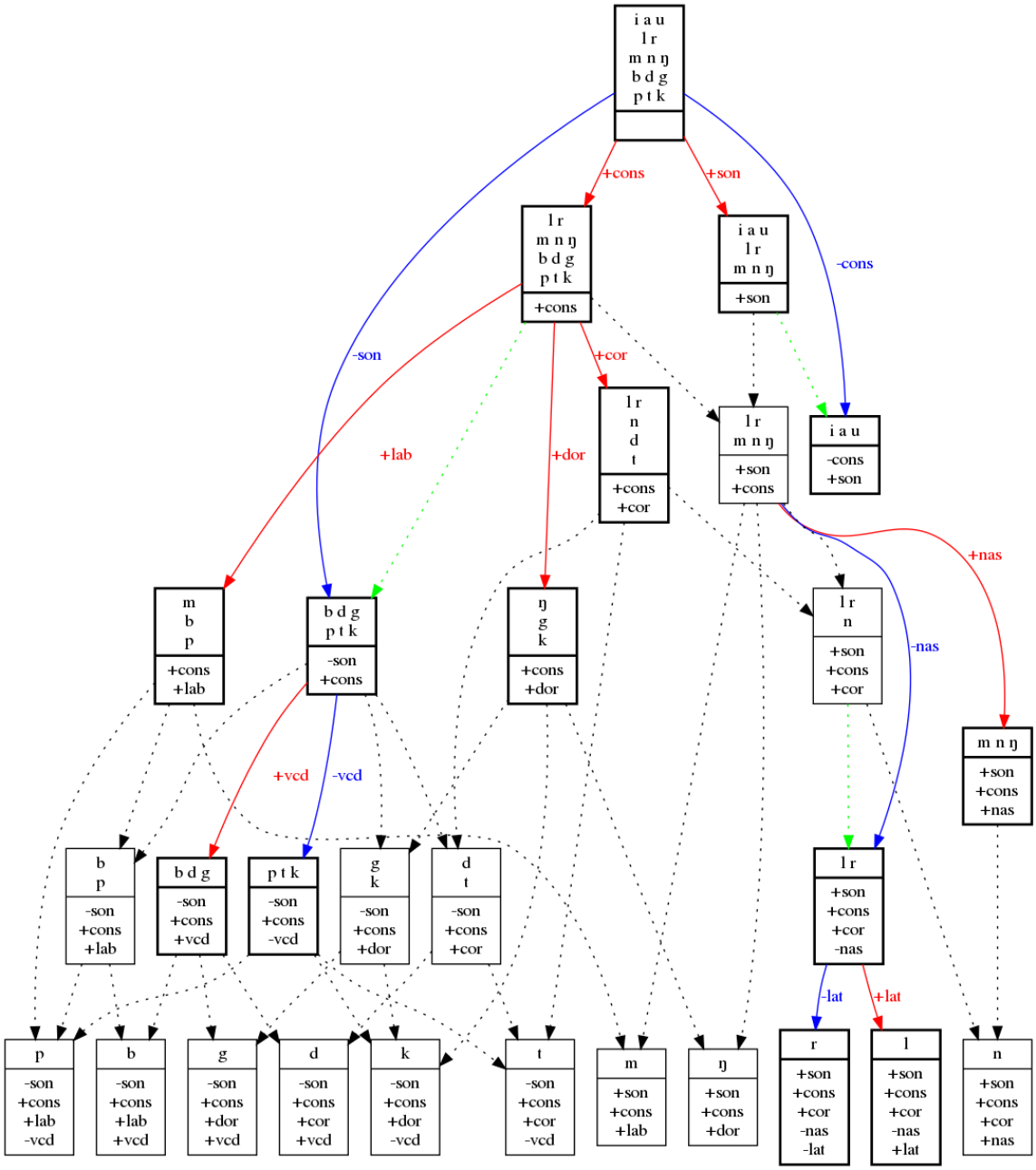

Figure 6: Big alphabet.