An algorithm to assign features to a set of phonological classes

Mayer, Connor
connormayer@ucla.edu

Daland, Robert
r.daland@gmail.com

Abstract

This squib describes a dynamic programming algorithm which assigns features to a set of phonological classes. The input consists of a set of classes, each containing one or more segments; in other words, a subset of the powerset of a segmental alphabet Σ. If a class can be generated as the union of existing features (= intersection of already-processed classes), those features are propagated to every segment in the class. Otherwise, a new feature/value pair is assigned. The algorithm comes in 4 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. The main text sets out necessary background, and illustrates each variant of the algorithm. The Appendix formally proves that each algorithm is sound.

1 Introduction

Distinctive features are the building blocks of phonological theory. Features represent phonetic qualities of speech sounds, and can be used in isolation or combination to describe individual sounds or classes of sounds (e.g., de Saussure, 1959; Jakobson, Gunnar, Fant, & Halle, 1952). This captures the generalization that sounds with similar phonetic properties tend to pattern similarly in the phonologies of languages. For example, the feature $\left[\ \text{-voice}\ \right]$ picks out all sounds without voicing, while the combination of features $\left[\begin{array}{c}\text{-voice}\\ \text{-continuant}\end{array}\right]$ picks out, for example, the English voiceless stop series.

Distinctive features have traditionally been described as *innate*: that is, all the sounds in the world's languages can be described by the same finite set of features, and the task of the language learner is to decompose the sounds of the ambient language into their consituent features to build a phonological grammar (e.g., Chomsky & Halle, 1968). This implies that the phonological classes we see in natural languages should be definable by combinations of these innate features.

Although feature theory has been extremely productive in phonology, there is evidence that shows many phonological classes cannot be described as a combination of phonetically-based features. For example, Mielke (2008) conducted a survey of almost 600 languages and showed that, at best, any modern feature system can categorize 71% of attested sound classes. The remaining 29% are *phonetically disparate classes*, which

require less theoretically-appealing descriptions combinations of features to be described, such as XOR feature classes (e.g., [-voice] or [-continuant] but not both).

The preponderance of these 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 regardless of whether they have any 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.

This sets the stage for the goals of the current paper, which are somewhat modest. The basic question we address is the inverse of how features have typically been approached: rather than asking what classes a feature system defines, we instead focus on how a feature system can be learned from a set of predetermined classes. We begin by defining a formal 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 calculation, we then describe a suite of algorithms for learning various types of featurizations for a set of input classes and prove their correctness, examining as we do the trade offs between number of classes and number of features that each featurization method makes.

This paper makes several important contributions: first, it demonstrates a method for working backwards to feature systems underpinning learned classes of sounds and provides the code for use in future research. Second, it provides a detailed formalization of what a featurization of classes entails, allowing careful reasoning about the expressiveness of such featurizations. Finally, by comparing multiple types of featurization of a set of classes, it makes explicit predictions about what classes should be describable under each type, which may be useful for future experimental phonological research.

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. Fig 1 illustrates this definition with a natural class system over a set of vowels.

In this graph, each node corresponds to a class, and a downward arrow from class X to class Y indicates that $Y \subset X$. However, the figure does not include an arrow for every pair of classes that have a subset relationship. For example, the class $\{\varepsilon\}$ is a subset of all vowels, but there is not an arrow from the class of all vowels to $\{\varepsilon\}$. This is because the class of front vowels {œ, y, ɛ, i} 'intervenes' between $\{\varepsilon\}$ and the class of all vowels.

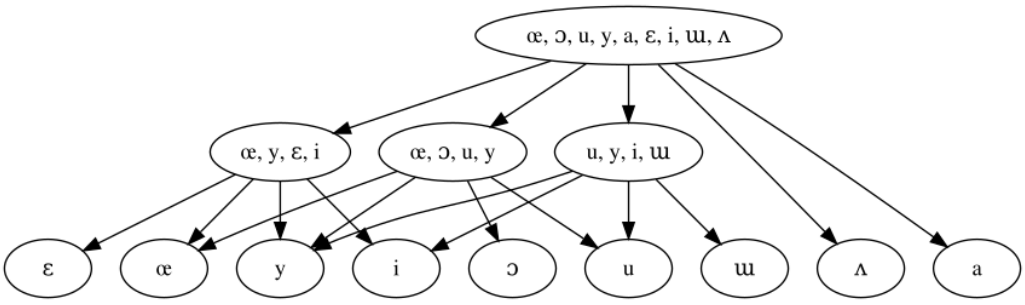

Figure 1: Vowel harmony lattice

We will use the terms *parent/daughter* to refer to cases in which a subset/superset relation holds and there is no intervening class.

Graphically, it is convenient to indicate parent-daughter relations in figures like 1, since other superset-subset relations are entailed. However, the parenthood relation is not only useful for graphing. As we show later, the parenthood relation is essential for the featurization algorithm. Thus, we formalize the definition here:

- X is a *parent* of Y (with respect to $\mathcal{C}$) if and only if $Y \subset X$, and $\nexists W \in \mathcal{C}\,[Y \subset W \subset X]$

We further define $\textsc{Parents}(Y, \mathcal{C})$ as the set of all parents of Y (with respect to $\mathcal{C}$).

2.2 Feature systems and featurizations

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 feature system for the vowel harmony lattice of Fig. 1 is shown below in Table 1. In the next subsection we formalize featural descriptors, which relate classes and feature systems.

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*: $\mathcal{V} = \{+, -\}$

σ	front	back	low	high	round
i	+	−	−	+	−
y	+	−	−	+	+
ɯ	−	+	−	+	−
u	−	+	−	+	+
ɛ	+	−	−	−	−
œ	+	−	−	−	+
ʌ	−	+	−	−	−
ɔ	−	+	−	−	+
a	−	+	+	−	−

Table 1: Example of a feature system.

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

A *featural descriptor* $\mathbf{e}$ is a set of feature/value pairs where the values cannot be 0, i.e. $\mathbf{e} \subset (\mathcal{V} \setminus \{0\}) \times \mathcal{F}$. For example, $\mathbf{e} = \left[\begin{array}{c} \text{+front} \\ \text{-low} \end{array}\right]$ is a featural descriptor.

To relate featural descriptors and phonological classes, note that every featural descriptor $\mathbf{e}$ can be expressed in the form $\mathbf{e} = \{\alpha_k F_k\}_{k=1}^{K}$, where each α_k is a value in $\mathcal{V} \setminus \{0\}$, and each F_k is some feature function $f_j \in \mathcal{F}$. Informally, we say that a featural descriptor describes the class of segments which have (at least) the feature/value pairs it contains. Formally, we write $\langle \mathbf{e} \rangle$ to indicate the natural class that corresponds to the featural descriptor $\mathbf{e}$:

$$\langle \{\alpha_k F_k\}_{k=1}^{K} \rangle = \{x \in \Sigma \mid F_k(x) = \alpha_k \text{ for every } k\}$$

We use the notation $\mathcal{V}^{\mathcal{F}}$ to denote the powerset of $(\mathcal{V} \setminus \{0\}) \times \mathcal{F}$, i.e. the set of all licit featural descriptors. Lastly, we define $\langle \mathcal{V}^{\mathcal{F}} \rangle = \{\langle \mathbf{e} \rangle \mid \mathbf{e} \in \mathcal{V}^{\mathcal{F}}\}$, the set of all natural classes described by some featural descriptor in $\mathcal{V}^{\mathcal{F}}$. We say that the feature system $(\mathcal{F}, \Sigma, \mathcal{V})$ generates the natural class system $\langle \mathcal{V}^{\mathcal{F}} \rangle$.

Note that while every featural descriptor in $\mathcal{V}^{\mathcal{F}}$ picks out a class in $\langle \mathcal{V}^{\mathcal{F}} \rangle$, the two are not 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 $\left[\begin{array}{c} \text{+front} \end{array}\right]$ picks out the same class as the featural descriptor $\left[\begin{array}{c} \text{+front} \\ \text{-low} \end{array}\right]$ (namely, the front vowels). Moreover, the featural descriptors $\left[\begin{array}{c} \text{+front} \\ \text{-front} \end{array}\right]$ and $\left[\begin{array}{c} \text{+high} \\ \text{+low} \end{array}\right]$ both pick out the empty set.

We say that a feature system $(\mathcal{F}, \Sigma, \mathcal{V})$ *covers* a natural class system $\mathcal{C}$ if $\mathcal{C} \subset \langle \mathcal{V}^{\mathcal{F}} \rangle$; in other words if the feature system provides a distinct featural representation for every

class in $\mathcal{C}$. In the remainder of this squib, we show how to construct a feature system that covers an arbitrary natural class system $\mathcal{C}$.

We begin with a worked-out example illustrating the difference between privative and full specification with the same segmental alphabet. Then we introduce the notion of intersectional closure, which leads naturally to a featurization algorithm for privative specification. Simple modifications yield algorithms for contrastive and full specification.

2.4 Example

Let $\Sigma = \{\text{R, D, T}\}$. Informally, the reader may think of [R] as a sonorant, [D] as a voiced obstruent, and [T] as a voiceless obstruent; accordingly we use the feature names *son* and *vcd*. In this section, we illustrate the consequences of privative versus full specification, using featurizations that are isomorphic (that is, they match on the + values, and differ only as to whether the non-+ values are 0 or −). We begin with Table 2.

σ	son	vcd
R	+	+
D	0	+
T	0	0

Table 2: Sonorants and obstruents with privative specification.

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

- [ ] – {R, D, T}
- [+son] – {R}
- [+vcd] – {R, D}

Note that this featurization provides no featural descriptor that uniquely picks out the voiceless obstruent [T], no way to pick out the obstruents [T] and [D] to the exclusion of [R], and no way to pick out the voiced obstruent [D] without [R].

Next, consider the isomorphic featurization in which the 0's from Table 2 are replaced with −'s:

σ	son	vcd
R	+	+
D	−	+
T	−	−

Table 3: Sonorants and obstruents with full specification.

The set of natural classes this featurization describes is much larger, because the number of (extensionally distinct) featural descriptors is larger:

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

An important generalization emerges from comparing these featurizations: the more 0's in the featurization, the greater the number of distinct feature functions that will be required to cover the same natural class system. In one sense, privative specification is more complex, because it will normally involve more features. However, in another sense, it is simpler, because there are only + values to handle and because it will result in fewer natural classes. Therefore, we will treat privative specification first. Prior to this, we introduce the notion of intersectional closure – the data structure that proves useful for efficiently assigning a privative feature system.

3 Intersectional closure

In this section we define the *intersectional closure* of a natural class system $\mathcal{C}$ as the set of classes that can be generated by intersecting Σ with any set 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 Definition

A collection of sets $\mathcal{C}$ is *intersectionally closed* if and only if $\forall(X, Y \in \mathcal{C})\,[X \cap Y \in \mathcal{C}]$.

We write $\mathcal{C}_\cap$ to indicate the *intersectional closure* of a natural class system $(\mathcal{C}, \Sigma)$. This the smallest intersectionally closed collection which contains Σ and every set in $\mathcal{C}$; in other words, the intersectional closure does not contain any classes except Σ and those which can be generated by finite intersections of classes from $\mathcal{C}$.

3.2 Feature systems generate an intersectional closure

Now we explain why any feature system that covers $\mathcal{C}$ must cover $\mathcal{C}_\cap$. The explanation rides on the dual relationship between featural descriptors and the classes they describe: the class described by the union of two featural descriptors is the intersection of the classes described by each of the descriptors alone. This principle can be illustrated with the following example, using the vowel system in Fig. 1. Let $\mathbf{e}_1 = \left[\text{ +front } \right]$ and $\mathbf{e}_2 = \left[\text{ +round } \right]$. Then

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

In other words, 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. Featural Intersection Lemma proves that this kind of relationship holds for any pair of featural descriptors (and the classes they describe).

Featural Intersection Lemma

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

A formal proof is included in the Appendix. The key utility of this Lemma is that it can be applied inductively, to relate the union of multiple featural descriptors with the intersection of multiple classes.

Intersectional Closure Covering Theorem

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

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, which we will omit hereafter) such that and $Y = \bigcap_i X_i$. The hypothesis that $\mathcal{C} \subset \langle \mathcal{V}^\mathcal{F} \rangle$ 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 $Y = \langle \bigcup_i \mathbf{e}_i \rangle$:

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

$= \langle \bigcup_i \mathbf{e}_i \rangle$

The preceding chain of logic demonstrates that if a class can be expressed as the intersection of natural 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. This completes the proof.

3.3 An algorithm for calculating the intersectional closure

The following algorithm yields the intersectional closure of a natural class system $(\mathcal{C}, \Sigma)$. A proof by induction 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 natural 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 arises in the intersectional closure of the vowel system in Fig. 1, shown in Fig. 2.

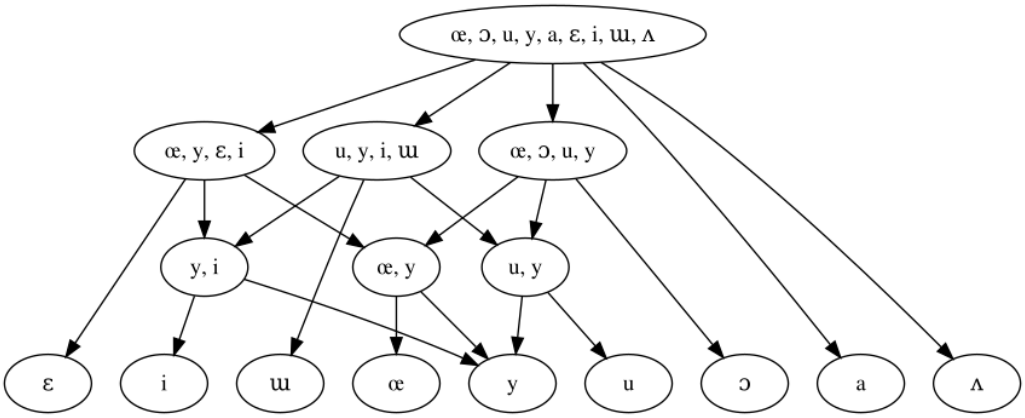

Figure 2: Intersectional closure of the vowel system shown earlier

In this case, the largest daughters of Σ are the $\left[\ +\text{front}\ \right]$ vowels, the $\left[\ +\text{high}\ \right]$ vowels, and the $\left[\ +\text{round}\ \right]$ vowels. The pairwise intersections of these classes give rise to the $\begin{bmatrix} +\text{front} \\ +\text{high} \end{bmatrix}$, $\begin{bmatrix} +\text{high} \\ +\text{round} \end{bmatrix}$, and $\begin{bmatrix} +\text{round} \\ +\text{front} \end{bmatrix}$ classes. The intersection of any pair of these is {y} (the high, front, round vowel), which accordingly has 3 parents. In the next section, we give an algorithm which generates a privative feature system $(\mathcal{F}, \Sigma, \{+, 0\})$ that covers the intersectional closure $\mathcal{C}_\cap$, given a natural class system $(\mathcal{C}, \Sigma)$.

4 Privative specification

The following algorithm yields a privative specification by assigning a different feature $\left[\ +\text{f}\ \right]$ to the segments in each class with a single parent.

Require: $\mathcal{C}_\cap$ is the intersectional closure of a natural 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 natural 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. 3, we illustrate the outcome of applying the privative specification algorithm to the intersectional closure of Fig. 2. We employ several conventions to jointly optimize readability and informativity. First, classes which have a single parent are visually highlighted using a thick, red ellipse. 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 that is used to distinguish it. This represents the 'point' at which each feature is assigned. Note that since the algorithm assigns feature/value pairs to *segments*, every daughter of the class which got a new feature also gets that feature. The set of features that is shared by all members of a class, i.e. which uniquely picks out that class and not any other, 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. And when the feature only picks out one segment, we name the feature after the segment. (But this is purely for readability of the figure. The algorithm knows nothing of phonological substance.)

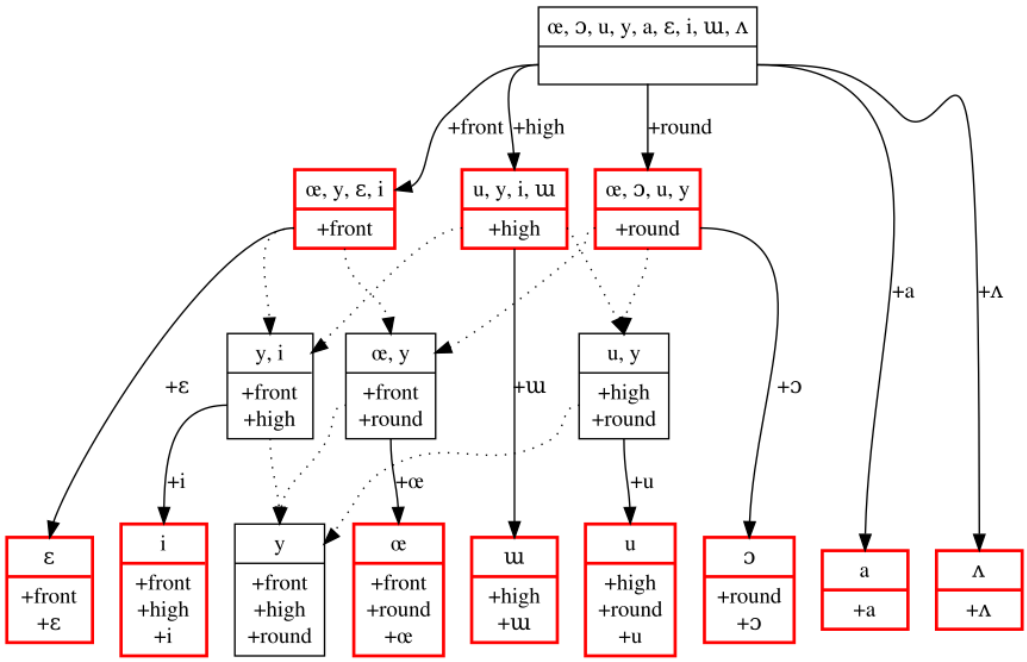

Figure 3: Yield of the privative specification algorithm

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 natural 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, besides those necessary ones. 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 wishing to allow '−' values. One might also wish to have an *economical* 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 economical feature system by 'adding' classes to the system. For example, the featurization shown in Fig. 3 contains 10 features (*front*, *high*, *round*, plus 7 features for the individual segments that cannot be accessed as combinations of front, high, and round). It is left as an exercise for the reader to verify that if the input consists of the following classes, the privative specification algorithm returns a featurization with 7 features:

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

Crucially, this featurization covers the original class system shown in Fig. 1. In other words, it uses less 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 privative specification algorithm returns a featurization in which the number of features is equal to the number of classes in the input with a single parent; this could be thought of as an upper bound. The second point this example makes is that adding the 'right' classes to the input is what enabled a more economical feature system. In the remainder of this paper, we explore variants of the privative specification algorithm which consider the complements of the input class (either with respect to the parent, or with respect to Σ) and assign '−' values instead of (or in addition to) '0' values.

5 Contrastive underspecification

One of the best cases for non-privative specifications arise from complement classes, such as round vs. nonround vowels, or voiced vs. voiceless obstruents. In a language with rounding harmony, like Turkish, one would need to write one harmony rule for the $\left[\ +\text{round}\ \right]$

feature, and a formally 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 characterize only vowels as [+round] or [-round]; in some languages, one might wish to only characterize non-low (or high) vowels only. In these 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 the relevant domain for complementation is with respect to the parent. And that is the change we introduce for contrastive underspecification: a '$-$' value is assigned when the complement of the class being processed (with respect to its parent) is in the input. We use the notation $\overline{X}$ to indicate the complement of X (with respect to a single parent). A formal specification of the algorithm follows.

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**
    $P_X \leftarrow \textsc{dequeue}(\textsc{parents}(X))$

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

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

    $\textsc{append}(\mathcal{F}, f_X)$
  end if
end while

The soundness of this algorithm follows from the soundness of the privative specification

algorithm. That is because the contrastive underspecification algorithm yields a feature system which generates the same class system as privative specification. The only 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 3-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 4: Sonorants and obstruents with contrastive underspecification.

The term contrastive underspecification is meant to capture that features can be contrastive, but segments can be forced to be underspecified with respect to a feature by omitting the relevant complement class from the input. For example, in Table 4 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 wasn't 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 classes to the covering. Every complement gets a '−' feature, including those which were not in the input. Because of this, it is necessary to update the intersectional closure. This can be done dynamically, by running the same intersectional closure algorithm as above, except starting with the pre-existing closure, and a queue consisting of the

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**
    $P_X \leftarrow \textsc{dequeue}(\textsc{parents}(X))$
    $\overline{X} \leftarrow P_X \setminus X$
    define $f_X : \Sigma \to \mathcal{V}$ by $f_X(\sigma) = \begin{cases} + & \text{if } \sigma \in X \\ - & \text{if } \sigma \in \overline{X} \\ 0 & \text{otherwise} \end{cases}$
    $\textsc{append}(\mathcal{F}, f_X)$

    $\mathcal{C}_\cap \leftarrow \textsc{IntersectionalClosure}(\mathcal{C}_\cap, \mathcal{Q} = \{\overline{X}\})$
  end if
end while

This algorithm is sound because it considers all the classes that the privative specification algorithm does, plus some more. Thus, it necessarily covers $\mathcal{C}$.

This algorithm is illustrated with the same vowel system that we have been using throughout, i.e. where C includes the front vowels, the high vowels, the round vowels, and all singletons.
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. Zero values are assigned to daughters-of-daughters. For example, suppose that stridents are daughters of coronals, and coronals are daughters of Σ. Then contrastive specification will create a $\left[\text{ -coronal }\right]$ class (all noncoronals) and a $\left[\text{ -strident }\right]$ class; the latter class will include all coronal nonstridents (but will not include, e.g. labials). 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 variant yields full specification.

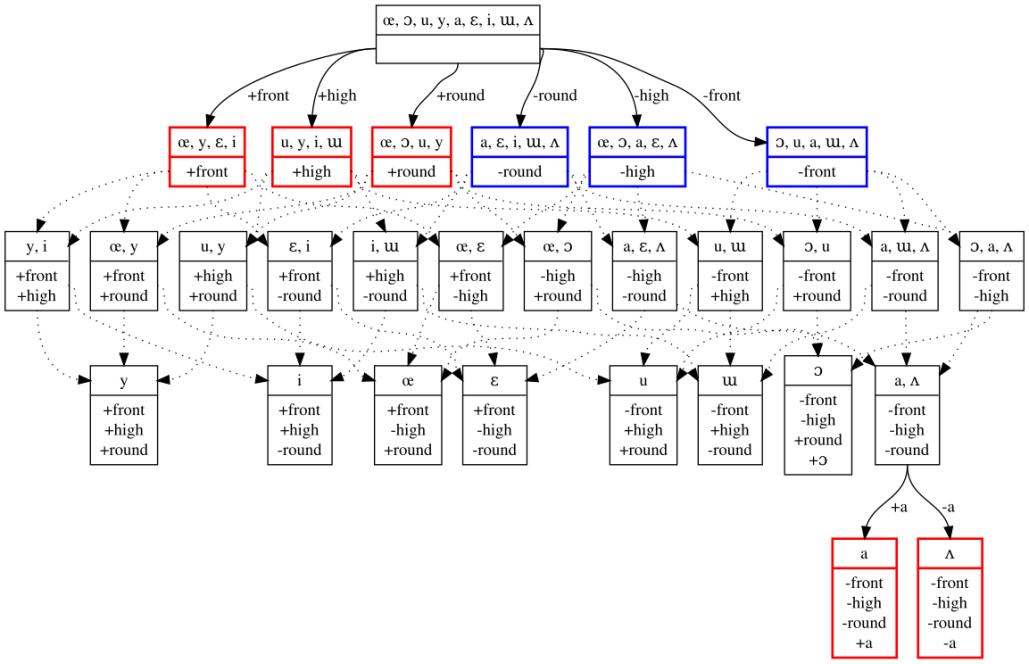

Figure 4: Class system and featurization yielded by contrastive specification

7 Full specification

achieved by assigning a new feature [+f] to every segment in X, and [-f] to every segment in $\Sigma \setminus X$

Require: $\mathcal{V} = \{+, -, 0\}$
Require: Precompute $C_\cap \leftarrow \textsc{IntersectionalClosure}(C)$
Ensure: Output featurization $\mathcal{F} = \{f_j\}_{j=1}^{M}$ such that $(\mathcal{F}, \Sigma, \mathcal{V})$ covers $\mathcal{C}$

$\mathcal{Q} \leftarrow \{c \in C_\cap \mid \|\textsc{Parents}(c)\| = 1\}$
$i \leftarrow 1$

while $\mathcal{Q} \neq \varnothing$ **do**
  $c \leftarrow \textsc{Dequeue}(\mathcal{Q})$
  $c' \leftarrow \Sigma \setminus c$

$$f_i(x) = \begin{cases} + & \text{if } x \in c \\ - & \text{if } x \in c' \\ 0 & \text{otherwise} \end{cases}$$

$C_\cap \leftarrow \textsc{IntersectionalClosure}(C_\cap \cup \{c'\})$

$\mathcal{Q} \leftarrow \{c_l \in \mathcal{Q} \mid c_l \neq c' \ \wedge \|\textsc{Parents}(c_l)\| = 1\}$

$\mathcal{F} \leftarrow \mathcal{F} \cup f_i$

$i \leftarrow i + 1$

end while

.1 Proof of Featural Intersection Lemma

The proof proceeds by showing that $\langle \mathbf{e}_i \cup \mathbf{e}_j \rangle \subset \langle \mathbf{e}_i \rangle \cap \langle \mathbf{e}_j \rangle$ and $\langle \mathbf{e}_i \rangle \cap \langle \mathbf{e}_j \rangle \subset \langle \mathbf{e}_i \cup \mathbf{e}_j \rangle$. 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.

.2 Proof of intersectional closure algorithm

We will show that the algorithm generates every class in the intersectional closure by induction. $\mathcal{C}_\cap$ 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.

.2.1 Representing subset relations as matrices

Any directed graph like Fig 1 has an equivalent representation with a square matrix. Row i is identified with class C_i, column j with C_j; and the value in the (i, j) cell indicates the

relationship between C_i and C_j. For binary relations like the subsethood and daughterhood, it is convenient to use the Boolean field: the set $\{T, F\}$ with operations $\cdot$ (logical AND), $+$ (logical OR), and additive inverse $-$ (logical NOT). The *subset matrix* is defined thus:

$$S_{ij} = \begin{cases} T & \text{if } C_j \subset C_i \\ F & \text{otherwise} \end{cases}$$

The advantage of writing a relation in this way is that fundamental matrix operations correspond to various measures of interest. For example, when S is defined as above, and S^2 is calculated from ordinary matrix multiplication, then S^2 indicates the subset-of-a-subset relation, which corresponds graphically to 'paths' of length 2 on a subset graph. In other words, $(S^2)_{ij} = T$ means there is a C_k such that $S_{ik} = T$ AND $S_{kj} = T$. If we further let AND denote the element-wise multiplication operator, then the *daughter matrix* can be computed from the subset matrix as follows:

$$D = S \text{ AND } -S^2$$

This matrix expresses the daughterhood relation: $D_{ij} = T$ means C_j is a daughter of C_i.[1] The daughterhood relation is of central importance to the featurization algorithms described in this paper.

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.

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

de Saussure, F. (1959). *Course in general linguistics.* New York: Philosophical Library.

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

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

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

[1] Similarly, the subset matrix can be expressed as the sum of all powers of the daughter matrix, $S = \sum_{n=1}^{N} D^n$. For this problem, we generally expect to compute D from S, rather than *vice versa*.