# An algorithm to assign features to a set of phonological classes

Mayer, Connor
connormayer@ucla.edu

Daland, Robert
r.daland@gmail.com

**Abstract**

This paper 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 $\Sigma$. 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. Otherwise, a new feature/value pair is assigned. 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.

## 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., de Saussure, 1959; Jakobson, Gunnar, Fant, & Halle, 1952).

The goal of feature theory is to capture the generalization that segments which are made alike tend to behave alike. For example, the English voiceless stops [p, t, $\widehat{\text{tʃ}}$, k] are all produced with a complete, long-lag closure of the oral cavity, 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. One underpinning of feature theory is typological: segments which are made alike (in different languages) tend to behave alike (in different languages). For example, the feature $\begin{bmatrix} \text{voice} \end{bmatrix}$ is posited to explain the fact that in many languages, all obstruents undergo the same voicing processes (regressive voicing assimilation within obstruent clusters; word-final devoicing; intervocalic and/or postnasal voicing, etc.).

For this reason, 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. Speakers inherently produce and perceive speech in terms of these features – they are the substantive 'atoms' of which segments and higher prosodic

constituents are composed. These texts assume that 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 described by a set of shared features. One well-trodden example is the *ruki* rule of Sanskrit, in which an underlying /s/ becomes retroflexed when it occurs after any of {r, u, k, i} (?, ?). While it has been proposed that the *ruki* process originated from the distinctive (though not uniform) effects of these segments on the second and third formant of neighboring segments (?, ?), it is widely agreed that no conventional feature system can pick out all four of these segments to the exclusion of others (?, ?). The issue is not limited to this one example. Mielke (2008) conducted a survey of phonological processes in almost 600 languages. In the 'best' feature system he considered, 71% of the classes which underwent or conditioned a phonological process could be expressed as the combination of simple features. The remaining 29% are *phonetically disparate classes* like {r, u, k, i}. Such classes require additional theoretical mechanisms – such as building classes through an OR operation – which seriously compromise the explanatory power that made feature theory appealing in the first place.

The 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 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. Finally we discuss some implication for feature theory and feature learning.

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[1]. Second, it provides a detailed formalization of what a

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

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 $\Sigma$ denote an alphabet of segments. We will use the term *class* to mean a subset of $\Sigma$.

### 2.1 Classes and class systems

A *class system* $(\mathcal{C}, \Sigma)$ consists of an alphabet $\Sigma$ 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 (the empty set is omitted in this and all similar figures).

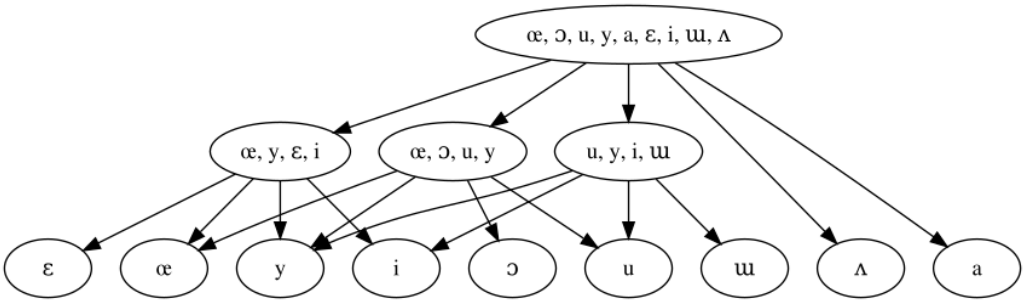


Figure 1: An example vowel system, used throughout this paper

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 {ɛ} is a subset of all vowels, but there is not an arrow from the class of all vowels to {ɛ}. This is because the class of front vowels {œ, y, ɛ, i} 'intervenes' between {ɛ} and the class of all vowels. 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 only parent-daughter relations in figures, 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

- $\Sigma$ 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 vowel system of Fig. 1 is shown below in Table 1. In the next subsection we formalize featural descriptors, which relate classes and feature systems.

| $\sigma$ | front | low | high | round |
|---|---|---|---|---|
| i | + | − | + | − |
| y | + | − | + | + |
| ɯ | − | − | + | − |
| u | − | − | + | + |
| ɛ | + | − | − | − |
| œ | + | − | − | + |
| ʌ | − | − | − | − |
| ɔ | − | − | − | + |
| a | − | + | − | − |

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

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, $\mathbf{e} = \begin{bmatrix} \text{+front} \\ \text{-low} \end{bmatrix}$ 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 $\alpha_k$ 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 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 $\begin{bmatrix} \text{+front} \end{bmatrix}$ picks out the same class as the featural descriptor $\begin{bmatrix} \text{+front} \\ \text{-low} \end{bmatrix}$ (namely, the front vowels). Moreover, the featural descriptors $\begin{bmatrix} \text{+front} \\ \text{-front} \end{bmatrix}$ and $\begin{bmatrix} \text{+high} \\ \text{+low} \end{bmatrix}$ 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 paper, 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 $\Sigma$ = {R, D, T} and $\mathcal{C}$ = {{R, D},{R}}. Informally, the reader may think of [R] as a sonorant, [D] as a voiced obstruent, and [T] as a voiceless obstruent and the classes {R, D} and {R} as voiced sounds and sonorants respectively; accordingly we use the feature names *vcd* and *son*. In this section, we illustrate the consequences of privative versus full specification, using featurizations which match on the '+' values, and differ only as to whether the other values are '0' or '−'. We begin with Table 2.
The set of natural classes it describes, and the simplest featural descriptor for each, are shown below:

- [ ] – {R, D, T}
- [+son] – {R}

| $\sigma$ | son | vcd |
|---|---|---|
| R | + | + |
| D | 0 | + |
| T | 0 | 0 |

Table 2: Sonorants and obstruents with privative specification.

- [+vcd] – {R, D}

Note that because featural descriptors cannot have values of '0', 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 featurization in which the '0's from Table 2 are replaced with '−'s:

| $\sigma$ | 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:

- [ ] = {R, D, T}
- [+son] = {R}
- [−son] = {D, T}
- [+vcd] = {R, D}
- [−vcd] = {T}
- [−son, +vcd] = {D}
- [+son, −vcd] = ∅

An important generalization emerges from comparing these featurizations: the more '0's in the featurization, the fewer classes a feature system will be able to describe. Correspondingly, a greater the number of distinct feature functions will be required to cover the same natural class system. A privative specification of the classes defined by the featurization in Table 3 (i.e. if we let $\mathcal{C}$ = {{D,T}, {R,D}, {T}, {D}, {R}}) would require four feature functions, where the two additional functions correspond to the '–' values of *son* and *vcd*.

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 feature systems.

# 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 $\Sigma$ 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 is the smallest intersectionally closed collection which contains $\Sigma$ and every set in $\mathcal{C}$. In other words, the intersectional closure does not contain any classes except $\Sigma$ 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

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

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. The 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 $\langle \mathbf{e}_i \cup \mathbf{e}_j \rangle = \langle \mathbf{e}_i \rangle \cap \langle \mathbf{e}_j \rangle$.

*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. This completes the proof.

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 $I$, hereafter omitted) 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 $Y = \langle \bigcup_i \mathbf{e}_i \rangle$:

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

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 $(\mathcal{C}, \Sigma)$.

**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**

*Proof of soundness*:

The proof goes by induction. First, we show that every class which can be generated by the intersection of 0 classes ($\Sigma$) 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 $\Sigma$. 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 ($\Sigma$) 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.

### 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 $X_1, X_2 \in \textsc{parents}(Y)$, 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 $\Sigma$, 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.

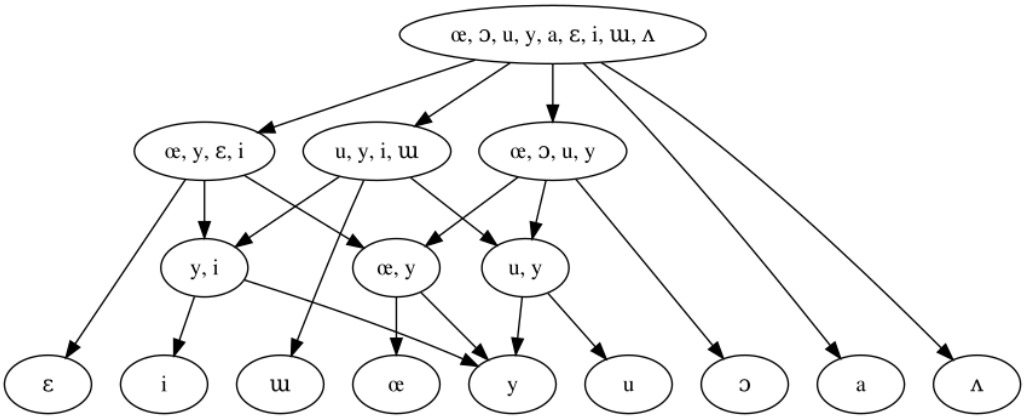


Figure 2: Intersectional closure of the vowel system shown earlier

In this case, the largest daughters of $\Sigma$ 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), and the intersection of all 3 is also {y}. Thus, this set has 3 parents, but the segments it contains are determined simply by having more than 1 parent.

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 $\left[\ +\text{f}\ \right]$ 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 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 $\Sigma$, 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 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. 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 (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.

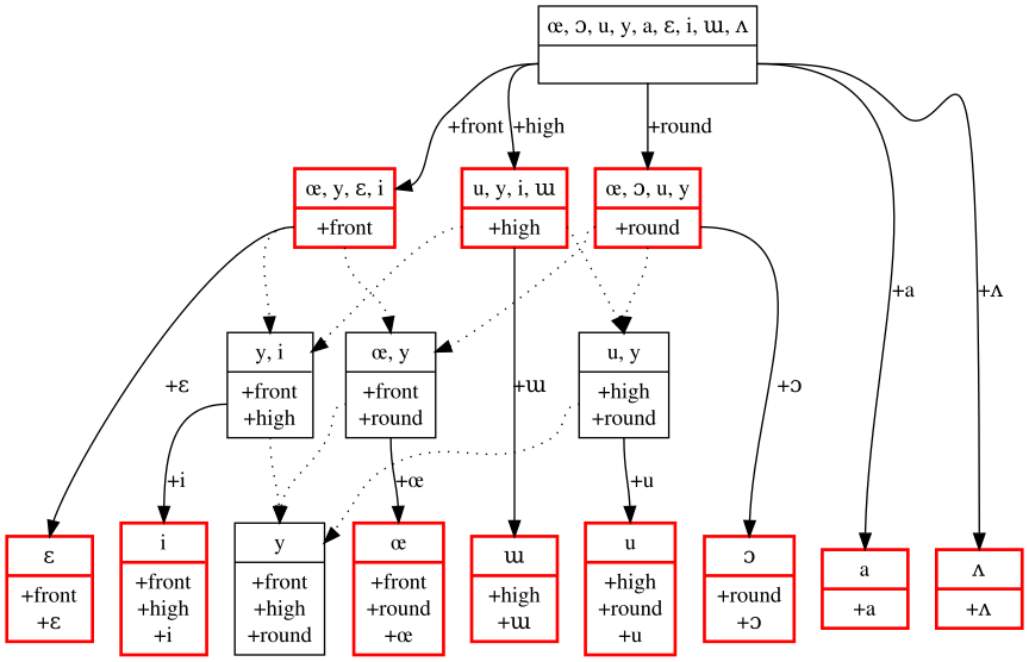


Figure 3: Yield of the privative specification algorithm

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 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 economy 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 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 [ +round ] feature, and an otherwise identical rule for the [ +nonround ] feature. By allowing features to take on opposing values, one formally recognizes the sameness of rounding with respect to the harmony process.

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

The central insight developed in this paper is that a new feature needs to be assigned just in case a class has a single parent. This suggests that the 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**
    $P_X \leftarrow \textsc{dequeue}(\textsc{parents}(X))$

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

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$(\mathcal{F}, f_X)$

**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 does. 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):

| $\sigma$ | 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 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 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, the intersectional closure actually changes throughout the computation. It can be updated dynamically, by running the intersectional closure algorithm, except starting with the pre-existing closure, and a queue consisting of the novel complement class. An additional important difference between this featurization and the previous two is that the ordering in which classes in $\mathcal{C}_\cap$ are processed is now crucial: in order to avoid adding spurious features, $\mathcal{C}_\cap$ must be processed using breadth-first search (i.e. each class' sisters are processed before its children). This ensures that no spurious features are added.

The algorithm proceeds in two steps: first, the complement classes are calculated during a breadth-first traversal of $\mathcal{C}_\cap$ and added. This separation of tasks is not strictly necessary, but simplifies things somewhat.

The algorithm for adding complement classes (AddComplementsContrastive) and a proof of its correctness are included in Appendix A. The algorithm for featurizing the resulting classes is shown below.

**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** $|$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$(\mathcal{F}, f_X)$
    remove$(\mathcal{Q}, \overline{X})$
  **end if**
**end while**

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

algorithm does, plus others. 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.

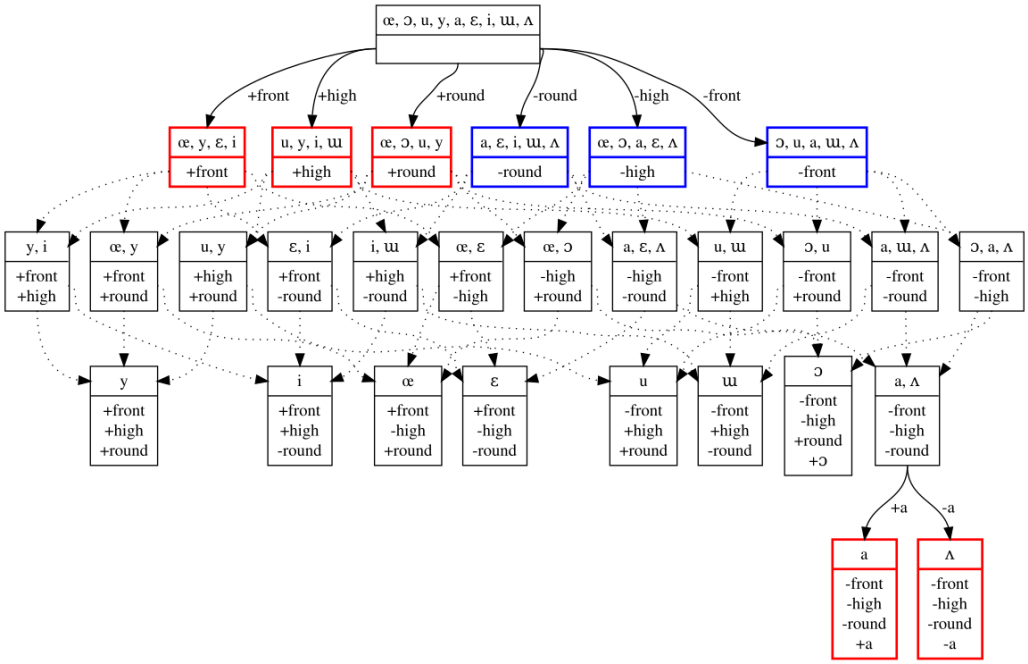


Figure 4: Class system and featurization yielded by contrastive specification

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, non-zero 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 $\Sigma$. Then contrastive specification will create a [ -coronal ] class (all noncoronals) and a [ -strident ] 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 $\Sigma$ rather than the single parent. Indeed, that is the final variant we discuss – full 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 added first, and then features are determined. This is described in Appendix A.

**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$(\mathcal{F}, f_X)$
    remove$(\mathcal{Q}, \overline{X})$
  **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. Fortunately, it can be avoided by ensuring that the nasals are generated by 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 is not generated

as a complement class (although, the set of continuant non-sonorants is, as well as the set of continuant sonorants and the set of non-continuant non-sonorants).

# 8 Discussion

In this paper, we have given 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 the 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 economy and expressivity

Here we present some examples which illustrate a little more about the expressiveness of class systems, and the relation between algorithm conservativeness and expressiveness.

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 $\Sigma$, and this is because each segment gets its own feature. This featurization will only generate the classes in the input (and $\Sigma$, 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 left as an exercise for the reader to show that when the privative specification algorithm is given this input, it generates the full powerset of $\Sigma$ – every possible subset gets a unique combination of features. Thus, privative specification is still compatible with a maximally expressive system.

The powerset of $\Sigma$ 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. 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 $\Sigma$ are included, the privative specification algorithm yields the same class system as the full specification one does (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, 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 are also concerned with the question of what 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 the same with respect to a harmony or assimilation process.

We do not at present have strong feelings regarding what a feature system *should* do. But we have something else – the received wisdom of the field. Our sense is that modern phonologists generally

- 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.

We briefly discuss the conditions necessary for assigning each type of feature under the contrastive underspecification algorithm, and then illustrate with a 'simple' example, which is nonetheless more complex than what we have introduced before.

- 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, $\Sigma$, 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 some ancestor other than $\Sigma$ is in the input

With these points in hand, 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. **??**. The features [ cons ] and [ son ] are binary because each one partitions $\Sigma$. 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 $\Sigma$) are included in the input.

TODO: make that fig!

We leave it as an exercise to the reader to investigate what happens to the 'voicing' feature if the input includes the class of all phonetically voiced segments (i.e. $\Sigma \setminus \{p, t, k\}$).

Finally, 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 investigations that we think could be testable in phonological experiments: e.g. are speakers able to infer the existence of productive phonological classes that are not present in the input but arise as a consequence of the featurization of the input classes? If so, under what conditions?

### 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 (?, ?). However, 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 Mayer (in progress) 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, Calvez, Nadal, & Dupoux, 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 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. 2. 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. 4, 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 $\Sigma$) 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**
  $X \leftarrow \textsc{dequeue}(\mathcal{Q})$
  $C \leftarrow \textsc{SortLargeToSmall}(\textsc{children}(X))$
  $C' \leftarrow \varnothing$
  **while** $C \neq \varnothing$ **do**
    $c \leftarrow \textsc{dequeue}(C)$
    **if** $|\textsc{parents}(c)| = 1 \wedge c \notin D$ **then**
      $\overline{c} \leftarrow X \setminus c$
      $\mathcal{C}_\cap \leftarrow \textsc{IntersectionalClosure}(\mathcal{C}_\cap, \mathcal{Q}' = \{\overline{c}\})$
      **for** $c' \in C$ **do**
        **if** $c' \subset \overline{c}$ **then**
          $\textsc{remove}(C, c')$
        **end if**

```
            APPEND(C', c̄)
            APPEND(D, c̄)
          end for
          APPEND(C', c)
          APPEND(D, c)
        end if
      end while
      ENQUEUE(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 $\Sigma$ 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.

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.

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.

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

Mayer, C. (in progress). Learning phonological features from distributional information.

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.