

# 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  $\Sigma$ . 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.,  $?$ ,  $?$ ,  $?$ ). This captures the generalization that sounds with similar phonetic properties tend to pattern similarly in the phonologies of languages. For example, the feature  $[-\text{voice}]$  picks out all sounds without voicing, while the combination of features  $\begin{bmatrix} -\text{voice} \\ -\text{continuant} \end{bmatrix}$  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 constituent features to build a phonological grammar (e.g.,  $?$ ,  $?$ ). 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,  $?$  ( $?$ ) 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., ?, ?, ?, ?, ?): 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  $\Sigma$  denote an alphabet of segments. We will use the term *class* to mean a subset of  $\Sigma$ . We will also use the notation  $\mathcal{P}(X)$  to indicate the *powerset* of  $X$  – the set of all subsets of  $X$ , including the empty set  $\emptyset$  and  $X$  itself.

### 2.1 Classes and class systems

A *class system*  $\mathcal{C}$  over an alphabet  $\Sigma$  is any subset of  $\mathcal{P}(\Sigma)$  which includes  $\Sigma$  itself. Formally,  $\mathcal{C} = \{C_i\}_{i=1}^N$ , where each  $C_i \subset \Sigma$ . Fig 1 illustrates this definition with a class system over a set of vowels.

In this graph, each node corresponds to a class, and a downward arrow from class  $C_i$  to class  $C_j$  indicates that  $C_j \subset C_i$ . To be more precise, downward arrows indicate that there

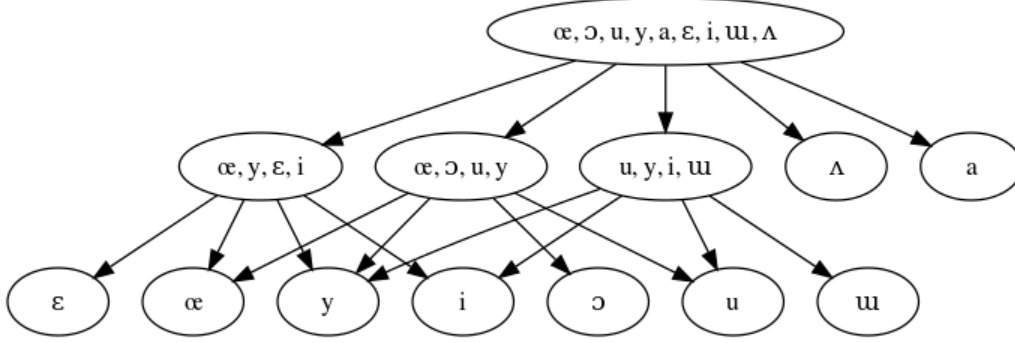


Figure 1: Vowel harmony lattice

is no ‘intervening’ class  $C_k$ . For example, the front vowels  $\{\alpha\epsilon, y, \epsilon, i\}$  intervene between all vowels and  $\{\epsilon\}$ , so we do not include an arrow from the top node to the  $\{\epsilon\}$  node. This relation will prove important for the algorithms we discuss later, and so we formalize it here:

**Definition 1**  $C_i$  is a parent of  $C_j$  (and  $C_j$  is a daughter of  $C_i$ ) if and only if  $C_j \subset C_i$ , and  $\nexists C_k \in \mathcal{C} [C_j \subset C_k \subset C_i]$ .

### 2.1.1 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 subethood 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 expressed the daughterhood relation:  $D_{ij} = T$  means  $C_j$  is a daughter of  $C_i$ .<sup>1</sup> The matrix  $D$  is of central importance to the featurization algorithms described later.

## 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  $f : \Sigma \rightarrow \mathcal{V}$  mapping segments to feature values

To illustrate, a feature system for the vowel harmony lattice shown in Fig. 1 is shown below:

$\sigma$	front	back	low	high	round
i	+	−	−	+	−
y	+	−	−	+	+
u	−	+	−	+	−
u	−	+	−	+	+
ε	+	−	−	−	−
œ	+	−	−	−	+
Λ	−	+	−	−	−
ɔ	−	+	−	−	+
a	−	+	+	−	−

Table 1: Example of a feature system.

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

---

<sup>1</sup>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*.

A *featural descriptor*  $\mathbf{e}$  is a set of feature/value pairs where the values cannot be 0,  $\mathbf{e} \in (\mathcal{V} \setminus \{0\}) \times \mathcal{F}$ . For example,  $\mathbf{e} = [+front, -low]$  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 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  $[+front]$  picks out the same class as the featural descriptor  $[+front, -low]$  (namely, the front vowels). Moreover, the featural descriptors  $[+front, -front]$  and  $[+high, +low]$  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  $\Sigma = \{R, D, T\}$ . 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.

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

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

Table 2: Sonorants and obstruents with privative specification.

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

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:

$\sigma$	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:

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

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  $\mathcal{C}$ . We prove 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 of a natural class system, as well as the intersection relation. It turns out that these structures are exactly what is needed to efficiently assign a feature system.

The *intersectional closure* of  $\mathcal{C}$ , denoted  $\mathcal{C}_\cap$ , is the natural class system consisting of every class that can be generated by the intersection of finitely many classes in  $\mathcal{C}$ . Formally,  $\mathcal{C}_\cap = \{\bigcap C' \in P \mid \exists P \in \mathcal{P}(\mathcal{C})\}$  (where  $\mathcal{P}(\mathcal{C})$  is the powerset of  $\mathcal{C}$ ). We show that if a feature system  $(\mathcal{F}, \Sigma, \mathcal{V})$  is rich enough to cover  $\mathcal{C}$ , it generates  $\mathcal{C}_\cap$ .

**Lemma:** 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  $\langle \mathbf{e}_i \cup \mathbf{e}_j \rangle \subset \langle \mathbf{e}_i \rangle \cap \langle \mathbf{e}_j \rangle$  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.

**Theorem:** Let  $\mathcal{C} = \{C_i\}_{i=1}^n$  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  $C$  be an arbitrary class in  $\mathcal{C}_\cap$ . By definition of  $\mathcal{C}_\cap$ , there exist  $\{C_i \in \mathcal{C}\}_{i=1}^n$  such that  $C = \bigcap_i C_i$ . The hypothesis that  $\mathcal{C} \subset \langle \mathcal{V}^\mathcal{F} \rangle$  implies that for every  $C_i$  in  $\mathcal{C}$ , there exists a featural descriptor  $\mathbf{e}_i$  such that  $\langle \mathbf{e}_i \rangle = C_i$ . Thus,  $C = \bigcap_i C_i = C_1 \cap C_2 \cap \dots \cap C_n$  can also be written  $C = \bigcap_i \langle \mathbf{e}_i \rangle = \langle \mathbf{e}_1 \rangle \cap \langle \mathbf{e}_2 \rangle \cap \dots \cap \langle \mathbf{e}_n \rangle$ . It follows by induction that  $C = \langle \bigcup_i \mathbf{e}_i \rangle$ :

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

$$\begin{aligned}
& \dots \\
& = \langle \mathbf{e}_1 \cup \mathbf{e}_2 \cup \dots \cup \mathbf{e}_n \rangle \\
& = \langle \bigcup_i \mathbf{e}_i \rangle
\end{aligned}$$

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. Thus, if  $(\mathcal{F}, \Sigma, \mathcal{V})$  covers  $\mathcal{C}$ , it covers the intersectional closure. This completes the proof.

Next, we give an algorithm which computes the intersectional closure, a modified variant of Dijkstra's shortest-paths algorithm. As we will show later, the computational benefit of precomputing the intersectional closure is that it efficiently computes the intersection relation, which reduces the computational complexity of the featurization algorithm. We assume that the input is a natural class system  $\mathcal{C} = \{C_i\}_{i=1}^N$ .

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

**Ensure:**  $\mathcal{I}(i, j) = k$  whenever  $C_i \cap C_j = C_k$  (for all  $i, j, k$ )

$\mathcal{C}_\cap \leftarrow \mathcal{C}$	{ordered list of classes}
$N \leftarrow  \mathcal{C}_\cap $	{size of $\mathcal{C}_\cap$ }
$\mathcal{Q} \leftarrow \{(i, j) \mid 1 \leq i, j \leq N, i \neq j\}$	{queue of natural class index pairs}
$\mathcal{I} \leftarrow \emptyset$	{hash table: $(i, j) \rightarrow k$ means $C_k = C_i \cap C_j$ }

```

while  $\mathcal{Q} \neq \emptyset$  do
   $(i, j) \leftarrow \text{POP}(\mathcal{Q})$ 
  if not  $(C_j \subset C_i \text{ or } C_i \subset C_j)$  then
     $C \leftarrow C_i \cap C_j$ 
    if  $C \in \mathcal{C}_\cap$  then
       $\mathcal{I}(i, j) \leftarrow \text{INDEX}(\mathcal{C}_\cap, C)$ 
    else
       $\text{APPEND}(\mathcal{C}_\cap, C)$ 
       $N \leftarrow N + 1$ 
      for  $k = 1$  to  $N$  do
         $\text{PUSH } (k, N) \rightarrow \mathcal{Q}$ 
         $\text{PUSH } (N, k) \rightarrow \mathcal{Q}$ 
      end for
    end if
  end if
end while

```



## 4 Privative specification

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

**Require:** Precompute  $C' < -\text{INTERSECTIONALCLOSURE}(C)$

**Require:**  $\mathcal{C}'$  is sorted in decreasing order of size ( $\forall i, j [i < j \rightarrow |C'_i| \geq |C'_j|]$ )

**Require:** parent matrix  $P$  ( $|C'| \times |C'|$ ):  $P_{ij} = 1$  if  $C'_j \subset C'_i \wedge \neg \exists k [C'_j \subset C'_k \subset C'_i]$ , 0 otherwise

**Ensure:** output featurization  $\mathcal{F} = \{f_j\}_{j=1}^M$  such that  $(\mathcal{F}, \Sigma, \mathcal{V})$  covers  $\mathcal{C}$

```
classQueue  $\leftarrow \{c_k \in C' \mid \sum_{j=1}^{|C'|} P_{jk} = 1\}$ 
 $i \leftarrow 1$ 
```

**while** classQueue  $\neq \emptyset$  **do**

```
   $c_k \leftarrow \text{POP}(\text{classQueue})$ 
```

```
   $f_i(x) = \begin{cases} + & \text{if } x \in c_k \\ 0 & \text{otherwise} \end{cases}$ 
```

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

```
   $i \leftarrow i + 1$ 
```

**end while**

The algorithm is considered sound if it returns a covering feature set for every valid input.

*Proof:*

## 5 Contrastive underspecification

achieved by assigning a new feature  $[+f]$  to every segment in  $X$ , and if  $Y \setminus X$  (the complement of  $X$  with respect to  $Y$ ) is in the input, then  $[-f]$  is assigned to every segment in  $Y \setminus X$

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

**Require:** Precompute  $C' < -\text{INTERSECTIONALCLOSURE}(C)$

**Require:**  $\mathcal{C}'$  is sorted in decreasing order of size ( $\forall i, j [i < j \rightarrow |C'_i| \geq |C'_j|]$ )

**Require:** parent matrix  $P$  ( $|C'| \times |C'|$ ):  $P_{ij} = 1$  if  $C'_j \subset C'_i \wedge \neg \exists k [C'_j \subset C'_k \subset C'_i]$ , 0 otherwise

**Ensure:** output featurization  $\mathcal{F} = \{f_j\}_{j=1}^M$  such that  $(\mathcal{F}, \Sigma, \mathcal{V})$  covers  $\mathcal{C}$

```

classQueue  $\leftarrow \{c_k \in C' \mid \sum_{j=1}^{|C'|} P_{jk} = 1\}$ 
 $i \leftarrow 1$ 

while classQueue  $\neq \emptyset$  do
   $c_k \leftarrow \text{POP}(\text{classQueue})$ 
  parent =  $\{c_i \in C \mid P_{ik} = 1\}$ 
   $c'_k \leftarrow \text{parent} \setminus c_k$ 
  if  $c'_k \in C$  then
     $f_i(x) = \begin{cases} + & \text{if } x \in c_k \\ - & \text{if } x \in c'_k \\ 0 & \text{otherwise} \end{cases}$ 
    classQueue  $\leftarrow \{x \in \text{classQueue} \mid x \neq c'_k\}$ 
  else
     $f_i(x) = \begin{cases} + & \text{if } x \in c_k \\ 0 & \text{otherwise} \end{cases}$ 
  end if
   $\mathcal{F} \leftarrow \mathcal{F} \cup f_i$ 
   $i \leftarrow i + 1$ 
end while

```

## 6 Contrastive specification

achieved by assigning a new feature [+f] to every segment in  $X$ , and [-f] to every segment in  $Y \setminus X$  (even if  $Y \setminus X$  was not in the input)

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

**Require:** Precompute  $C' < -\text{INTERSECTIONALCLOSURE}(C)$

**Require:**  $C'$  is sorted in decreasing order of size ( $\forall i, j [i < j \rightarrow |C'_i| \geq |C'_j|]$ )

**Require:** parent matrix  $P$  ( $|C'| \times |C'|$ ):  $P_{ij} = 1$  if  $C'_j \subset C'_i \wedge \neg \exists k [C'_j \subset C'_k \subset C'_i]$ , 0 otherwise

**Ensure:** output featurization  $\mathcal{F} = \{f_j\}_{j=1}^M$  such that  $(\mathcal{F}, \Sigma, \mathcal{V})$  covers  $\mathcal{C}$

```

classQueue  $\leftarrow \{c_k \in C' \mid \sum_{j=1}^{|C'|} P_{jk} = 1\}$ 
 $i \leftarrow 1$ 

```

```

while classQueue  $\neq \emptyset$  do
   $c_k \leftarrow \text{POP}(\text{classQueue})$ 
  parent =  $\{c_i \in C \mid P_{ik} = 1\}$ 

```

```

 $c'_k \leftarrow \text{parent} \setminus c_k$ 

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

 $C' \leftarrow (C' \cup \{c'_k\})$ 
RECALCULATE  $P$  from  $C'$ 
 $\text{classQueue} \leftarrow \{c_l \in \text{classQueue} \mid c_l \neq c'_k \wedge \sum_{j=1}^{|C'|} P_{jl} = 1\}$ 
SORT( $\text{classQueue}$ )
 $\mathcal{F} \leftarrow \mathcal{F} \cup f_i$ 
 $i \leftarrow i + 1$ 
end while

```

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

**Require:** Precompute  $C' < -\text{INTERSECTIONALCLOSURE}(C)$

**Require:**  $C'$  is sorted in decreasing order of size ( $\forall i, j [i < j \rightarrow |C'_i| \geq |C'_j|]$ )

**Require:** parent matrix  $P$  ( $|C'| \times |C'|$ ):  $P_{ij} = 1$  if  $C'_j \subset C'_i \wedge \neg \exists k [C'_j \subset C'_k \subset C'_i]$ , 0 otherwise

**Ensure:** output featurization  $\mathcal{F} = \{f_j\}_{j=1}^M$  such that  $(\mathcal{F}, \Sigma, \mathcal{V})$  covers  $\mathcal{C}$

```

 $\text{classQueue} \leftarrow \{c_k \in C' \mid \sum_{j=1}^{|C'|} P_{jk} = 1\}$ 
 $i \leftarrow 1$ 

while  $\text{classQueue} \neq \emptyset$  do
   $c_k \leftarrow \text{POP}(\text{classQueue})$ 
   $c'_k \leftarrow \Sigma \setminus c_k$ 
  
$$f_i(x) = \begin{cases} + & \text{if } x \in c_k \\ - & \text{if } x \in c'_k \end{cases}$$

   $C' \leftarrow (C' \cup \{c'_k\})$ 
  RECALCULATE  $P$  from  $C'$ 
   $\text{classQueue} \leftarrow \{c_l \in \text{classQueue} \mid c_l \neq c'_k \wedge \sum_{j=1}^{|C'|} P_{jl} = 1\}$ 
   $\mathcal{F} \leftarrow \mathcal{F} \cup f_i$ 
   $i \leftarrow i + 1$ 
end while

```

## A Formal proof of the algorithm

### A.1 Privative underspecification

### A.2 Contrastive underspecification

### A.3 Contrastive specification

### A.4 Full specification