Convexity of Neural Codes

R. Amzi Jeffs

Mohamed Omar, Advisor

Nora Youngs, Reader

Department of Mathematics

May, 2016

Abstract

An important task in neuroscience is stimulus reconstruction: given activity in the brain, what stimulus could have caused it? We build on previous literature which uses neural codes to approach this problem mathematically. A neural code is a collection of binary vectors that record concurrent firing of neurons in the brain. We consider neural codes arising from place cells, which are neurons that track an animal's position in space. We examine algebraic objects associated to neural codes, and completely characterize a certain class of maps between these objects. Furthermore, we show that such maps have natural geometric implications related to the receptive fields of place cells. Lastly we describe several purely geometric results related to neural codes.

Contents

Acknowledgments

I would like to thank my advisor Mohamed Omar for his guidance on this project. I would also like to thank my reader Nora Youngs for her extensive feedback and suggestions throughout the year. Both deserve gratitude for helping me navigate the existing literature in this field, and for their patience as I have sometimes veered from the beaten path. In addition I wish to thank Natchanon Suaysom and Aleina Wachtel for discussion regarding the algorithms and concepts in Section 2.5.

Finally, I owe thanks to my family for their unwavering support during my college career.

Chapter 1

Introduction and Preliminaries

In 1971 O'Keefe and Dostrovsky (1971) described spatial information encoded in the hippocampus of rats. The authors found that certain cells fired only in particular regions of a rat's environment, and that together these cells encoded a "spatial map" of the rat's surroundings. These neurons, known as place cells, play an important role in an animal's perception of space.

A critical problem that arises in the study of place cells is that of stimulus reconstruction. If we observe a collection of place cells firing in the brain, what can we infer about the spatial stimulus that caused this activity? In Curto and Itskov (2008) the authors addressed this question mathematically by considering combinatorial neural codes. A combinatorial neural code is a collection of binary vectors which encode concurrent firing behavior among neurons. The authors found that topological features of an animal's environment can be extracted directly from these codes. Recent work has sought to understand these codes and their associated stimuli from many perspectives, including algebraic geometry, topology, and combinatorics. These codes can be used to understand the behavior of place cells as well as other types of neurons. There exist other types of neural codes, but in this thesis we will consider only combinatorial neural codes.

For a given code we will assume that its associated stimulus come from a Euclidean topological space $X \subseteq \mathbb{R}^d$. We associate each place cell to the set of points in X where it fires, and our general task is stimulus reconstruction. Given a neural code, we want to associate each neuron to a set in X so that we obtain the original code as a response to the space. In the remainder of this chapter we will introduce and make precise the terminology used throughout.

1.1 Neural Codes and Realizations

Neural codes serve as a combinatorial representation of spatial information stored in the brain. As mentioned previously, these codes arise according to the firing patterns of neurons (specifically place cells). There is no inherent restriction on which codes are neural codes. Indeed, any collection of binary vectors with the same number of bits can be thought of as arising from the firing of neurons.

Definition 1.1 (Curto et al. (2013))**.** A *code* or *neural code* is a set $C \subseteq \{0,1\}^n$ of binary vectors. The vectors in C are called *codewords*.

Each vector in the code indicates a set of neurons that fire concurrently. For example, if a code C contains a codeword whose third and fourth bits are both 1 while all other bits are 0, this indicates that the third and fourth neuron fired concurrently while no other neurons were active. Each neuron can be associated naturally to the set of points in the stimulus space where it fires. Conversely, given a collection of sets one can associate each set to a neuron, and analyze the resulting code structure.

Definition 1.2 (Curto et al. (2013))**.** Let $\mathcal{U} = \{U_1, \ldots, U_n\}$ be a collection of sets in a topological space X. The *code of* $\mathcal{U}$ is the neural code defined by

$$C(\mathcal{U}) := \left\{ c \in \{0,1\}^n \,\middle|\, \left(\bigcap_{c_i=1} U_i\right) \setminus \left(\bigcup_{c_j=0} U_j\right) \neq \emptyset \right\}.$$

By convention, the empty intersection is the entire space X, while the empty union is the empty set. This means that if the sets in $\mathcal{U}$ do not cover X, we will include $00\cdots0$ in the code since $X \setminus \bigcup_{j\in[n]} U_j$ will be nonempty.

Often $\mathcal{U}$ will be a collection of open sets, but this is not always the case. It is sometimes informative to let the sets be closed, or place other constraints on them. As mentioned before, we will always assume that the space X can be embedded in Euclidean space of some dimension, so $X \subseteq \mathbb{R}^d$ for some d. Translating from a collection of sets to a code is at first opaque, and so considering an example is appropriate.

Figure 1.1 shows 5 open sets in $\mathbb{R}^2$. Note that since these sets do not cover all of $\mathbb{R}^2$, we include 00000 in our code. We can see that codewords in $C(\mathcal{U})$ are naturally associated to the different regions of intersection between the sets in $\mathcal{U}$. For example, 00110 is associated to the region $(U_3 \cap U_4) \setminus (U_1 \cup U_2 \cup U_5)$. This observation motivates the definition of a codeword region.

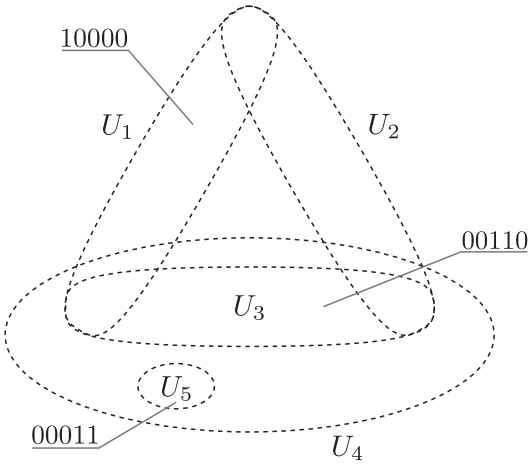


Figure 1.1 A collection $\mathcal{U} = \{U_1, U_2, U_3, U_4, U_5\}$ of sets in $\mathbb{R}^2$ with several codeword regions labeled. The full code of these sets is $C(\mathcal{U}) = \{00000, 10000, 01000, 00010, 11000, 10010, 01010, 00110, 00011, 10110, 01110\}$.

Definition 1.3 (Jeffs et al. (2015))**.** Let $\mathcal{U} = \{U_1, \ldots, U_n\}$ be a collection of sets and $v \in \{0,1\}^n$ be a binary vector. Then the *codeword region* of v in $\mathcal{U}$ is the set

$$\mathcal{U}(v) := \bigcap_{v_i=1} U_i \setminus \bigcup_{v_j=0} U_j.$$

Note that although codeword regions are defined in Jeffs et al. (2015) and discussed in Curto et al. (2013), our notation differs from previous literature slightly. The notation for the codeword region in Jeffs et al. (2015) is V_c for a vector c. Here we have chosen to write $\mathcal{U}(v)$ to make it clear that the codeword region can vary depending on the realization $\mathcal{U}$. This definition of a codeword region allows us to rewrite Definition 1.2 more compactly as $C(\mathcal{U}) = \{c \in \{0,1\}^n \mid \mathcal{U}(c) \neq \emptyset\}$.

It is straightforward to write down the code of a collection of sets $\mathcal{U}$, but can we go the other direction? Given a code, can we find a collection of sets that produce this code? We call such a collection of sets a realization.

Definition 1.4 (Curto et al. (2013))**.** Let $\mathcal{C}$ be a code. If $\mathcal{C} = C(\mathcal{U})$ for some collection of sets $\mathcal{U}$ in a space X, then we say that $\mathcal{U}$ is a *realization* of $\mathcal{C}$ in X.

As it turns out, all codes have a realization in some space $X \subseteq \mathbb{R}^d$ for any positive integer d. The following proposition proves this claim for $d = 1$,

and the same strategy works for any positive d.

Proposition 1.5 (Curto et al. (2013))**.** *Let C be a code. Then there exists a space $X \subseteq \mathbb{R}^1$ in which C has a realization.*

Proof. Index the codewords in C as $\{c_1, \ldots, c_m\}$ and let $B_1, \ldots, B_m$ be disjoint open intervals in $\mathbb{R}^1$. Set $U_i = \bigcup_{i \in \text{supp}(c_k)} B_k$, and $X = \bigcup_{j=1}^{m} B_j$. By construction, the collection $\mathcal{U} = \{U_1, \ldots, U_m\}$ is a realization of C in X. □

When we place restrictions on the space X and the sets in $\mathcal{U}$, the question of which codes have realizations becomes more interesting. Constraints on our space can model biological phenomena realistically, giving insight into which codes arise in natural settings. It is therefore important to classify how codes and their realizations behave under certain restrictions. Determining which codes have realizations consisting of convex open sets is one of the fundamental questions in the study of neural codes. The following section introduces these notions in more detail.

1.2 Convex Codes

Since neural codes arise biologically, it is natural to examine the firing regions of neurons in reality. As evidenced by Proposition 1.5, all codes have a realization in some subspace of $\mathbb{R}^d$. However, such realizations are not a realistic model of biological data. In fact, biological data suggests that the firing regions of place cells are typically convex, and at a minimum are connected.

Intuitively, a convex set is a subset of Euclidean space in which any point can be "seen" from any other point.

Definition 1.6. A *convex* set is a set $A \subseteq \mathbb{R}^d$ such that for any $p, q \in A$, the line segment from p to q is in A. More precisely, A is convex if and only if

$$\{(1-t)p + tq \mid 0 \leq t \leq 1\}$$

is a subset of A for all $p, q \in A$.

Figure 1.2 gives an example of a convex set and a nonconvex set.

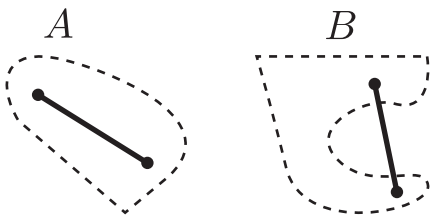


Figure 1.2 A convex set A and a nonconvex set B.

Definition 1.7. Let C be a code. If C has a realization consisting of open convex sets in a convex space $X \subseteq \mathbb{R}^d$ then C is called a *convex code*.

Convex codes realistically model many kinds of neural data, and classifying them precisely is difficult. Not all codes are convex, and the obstructions to convexity are numerous and varied. As a first example, the following proposition presents a code on three neurons which is never convex in any dimension.

Proposition 1.8 (Curto et al. (2013))**.** *Not all codes are convex. In particular,* $C = \{0,1\}^3 \setminus \{111, 100\}$ *does not have a realization consisting of convex open sets in any convex subspace of* $\mathbb{R}^d$*.*

Proof. Suppose that C did have some convex realization $\mathcal{U} = \{U_1, U_2, U_3\}$ in a convex space $X \subseteq \mathbb{R}^d$. Since 110 is in the code, there exists a point $p \in U_1 \cap U_2$ but not in U_3. Likewise, since 101 is in the code there exists a point $q \in U_1 \cap U_3$ but not in U_2. Now consider a line segment between p and q. Since both p and q are in U_1 and U_1 is convex, this line segment is contained entirely in U_1.

There are two cases: either U_2 and U_3 cover this line segment, or they do not. If the sets do not cover the line then there is some point on the line which is in *only* U_1. Then C must contain 100, a contradiction. Otherwise, U_2 and U_3 form an open cover of the line segment, which is a connected space. Thus U_2 and U_3 overlap at some point on the line. This point is then in $U_1 \cap U_2 \cap U_3$, and the code C must contain 111. Again this is a contradiction, and we conclude that C is not a convex code. □

Note that the code given in Figure 1.1 is a convex code with a realization in $\mathbb{R}^2$.

1.3 Combinatorial and Geometric Tools

So far we have treated codes simply as collections of binary vectors, but they can also be thought of as collections of subsets of $[n]$. In particular, we can associate each vector to the set of indices on which it is nonzero. This association provides a combinatorial perspective from which to view codes and makes certain characterizations more natural and elegant.

Definition 1.9 (Jeffs et al. (2015))**.** The *support* of a vector $v \in \{0,1\}^n$ is the set $\operatorname{supp}(v) := \{i \mid v_i \neq 0\}$ of indices at which the bits in v are nonzero. The

support of a code C is the collection containing the support of each vector in C:

$$\operatorname{supp}(C) := \{\operatorname{supp}(c) \mid c \in C\}.$$

Since a code and its support encode the same information we will use the two interchangeably when it does not give rise to ambiguity. Likewise, we will often avoid distinguishing between vectors on n bits and subsets of $[n]$. In some cases it can be useful to restrict a code to a smaller number of indices by "zeroing out" all other indices.

Definition 1.10. Let C be a code on n bits and let $\sigma \subseteq [n]$. Then the *restriction* of C to σ, denoted $C|_\sigma$, is the code on n bits obtained by setting bits whose indices are not in σ to zero. Equivalently,

$$C|_\sigma := \{\sigma \cap \tau \mid \tau \in C\}.$$

Restriction is the result of forgetting the behavior of all neurons except those in σ. The resulting code is still on n bits, but any realization of this code will necessarily have empty sets for all neurons not in σ. Hence the resulting code has at most $|\sigma|$ nontrivial bits.

Closely related to supports of codes are simplicial complexes, which are combinatorial objects with slightly more structure than the support of a code. Simplicial complexes have geometric properties which can often be used to infer information about a code.

Definition 1.11 (Curto et al. (2013))**.** A *simplicial complex* is a set $\Delta \subseteq 2^{[n]}$ which is closed under taking subsets. That is, if $\sigma \in \Delta$ and $\tau \subseteq \sigma$ then $\tau \in \Delta$. The sets in Δ are called *faces*. A *facet* of Δ is a face which is not properly contained in any other face of Δ. If Δ has a unique facet then we say that it is a *simplex*.

Observe that a simplicial complex is uniquely determined by its facets. The property that a simplicial complex is closed under taking subsets is sometimes referred to as being "downclosed." Given any collection S of subsets of $[n]$ we can downclose S to form a minimal simplicial complex containing S.

Definition 1.12 (Curto et al. (2013))**.** Let $S \subseteq 2^{[n]}$. Then the *simplicial complex* or *downclosure* of S is the set

$$\Delta(S) := \{\tau \subseteq [n] \mid \tau \subseteq \sigma \text{ for some } \sigma \in S\}.$$

One can verify that $\Delta(S)$ is indeed a simplicial complex, and that in fact it is the smallest simplicial complex containing S as a subset. When S is $\text{supp}(C)$ for some code C we will write $\Delta(C)$ for the simplicial complex of the support. Returning to the code given in Figure 1.1, we can write $C = C(\mathcal{U})$ in terms of its support as follows:

$$\begin{aligned} C &= \{00000, 10000, 01000, 00010, 11000, 10010, 01010, 00110, 00011, 10110, 01110\} \\ &= \{\emptyset, 1, 2, 4, 12, 14, 24, 34, 45, 134, 234\}. \end{aligned}$$

In the expression above we have omitted brackets and commas from sets in the support to avoid clutter. Taking the downclosure of C, we have that

$$\Delta(C) = \{\emptyset, 1, 2, 3, 4, 5, 12, 13, 14, 23, 24, 34, 45, 134, 234\}.$$

The facets of this simplicial complex are 12, 45, 134, and 234. Note that these correspond to the regions of highest order intersection in Figure 1.1. In this case we had to add several sets to C to obtain a simplicial complex, but this is not always the case since some codes have supports which are already simplicial complexes.

An important notion in simplicial complexes is the link of a face. Intuitively, the link of a face $\sigma \in \Delta$ is the result of localizing to sets containing σ. One can generalize the concept of a link to arbitrary codes. We provide both definitions below.

Definition 1.13. Let Δ be a simplicial complex and let $\sigma \in \Delta$ be a face of Δ. Then the *link* of σ in Δ is the simplicial complex

$$\text{Lk}_\sigma(\Delta) := \{\tau \subseteq [n] \mid \sigma \cap \tau = \emptyset \text{ and } \sigma \cup \tau \in \Delta\}.$$

Definition 1.14. Let C be a code on n bits and let $\sigma \subseteq [n]$ be any set. The *link* of σ in C is a code on n bits given by

$$\text{Lk}_\sigma(C) := \{\tau \subseteq [n] \mid \sigma \cap \tau = \emptyset \text{ and } \sigma \cup \tau \in C\}.$$

Notice that these definitions are compatible in the sense that if C is a simplicial complex then the link of any face σ in C as a code is the same as its link in C as a simplicial complex. It is also worth noting that $\text{Lk}_\sigma(C)$ need not be a subset of C (e.g., if σ is a subset of all sets in C).

One can think of the link as the result of collecting all sets in C containing σ, and removing σ from each set. This observation yields an intuitive characterization of the link, given below. It is also useful to observe that if C is a simplicial complex then so is any link in C.

Lemma 1.15. *Let C be a code on n bits and $\sigma \subseteq [n]$. Then*

(*i*) $\tau \in \mathrm{Lk}_\sigma(C)$ *if and only if* $\tau = \gamma \setminus \sigma$ *for some* $\gamma \in C$, *and*

(*ii*) *If C is a simplicial complex then so is* $\mathrm{Lk}_\sigma(C)$.

Proof. To see that the forward direction of (i) holds, let $\gamma = \sigma \cup \tau$. Then note that γ is in C by the definition of the link, and since σ and τ are disjoint, we have that $\tau = \gamma \setminus \sigma$. The reverse direction follows from the fact that if $\tau = \gamma \setminus \sigma$ for $\gamma \in C$, then τ is necessarily disjoint from σ and $\sigma \cup \tau = \gamma$ is in C.

For (ii), let τ be in $\mathrm{Lk}_\sigma(C)$ and let τ' be any subset of τ. Clearly τ' is disjoint from σ, and furthermore $\tau' \cup \sigma$ is a subset of $\tau \cup \sigma$, which is a face of C. Since C is downclosed, $\tau' \cup \sigma$ must also be in C, and so τ' is in $\mathrm{Lk}_\sigma(C)$. Thus $\mathrm{Lk}_\sigma(C)$ is downclosed as desired. □

Links provide a method of localizing within a code while preserving important structural properties. Furthermore, linking plays nicely with operations such as downclosing, union, and intersection, as illustrated by the following lemmas.

Lemma 1.16. *Linking commutes with downclosure. That is,* $\mathrm{Lk}_\sigma(\Delta(C)) = \Delta(\mathrm{Lk}_\sigma(C))$ *for any code C on n bits and $\sigma \subseteq [n]$.*

Proof. We can apply the definitions of linking and downclosure straightforwardly to compute that

$$\begin{aligned}
\mathrm{Lk}_\sigma(\Delta(C)) &= \{\tau \subseteq [n] \mid \sigma \cap \tau = \emptyset \text{ and } \sigma \cup \tau \in \Delta(C)\} \\
&= \{\tau \subseteq [n] \mid \sigma \cap \tau = \emptyset \text{ and } \sigma \cup \tau \subseteq \gamma \text{ for some } \gamma \in C\} \\
&= \{\tau \subseteq [n] \mid \tau \subseteq (\gamma \setminus \sigma) \text{ for some } \gamma \in C\} \\
&= \{\tau \subseteq [n] \mid \tau \subseteq \gamma' \text{ for some } \gamma' \in \mathrm{Lk}_\sigma(C)\} \\
&= \Delta(\mathrm{Lk}_\sigma(C)).
\end{aligned}$$

Note that the second to last equality follows from (i) of Lemma 1.15. This proves the desired result. □

Lemma 1.17. *Let Δ be a simplicial complex on n vertices and let $\sigma_1, \sigma_2 \subseteq [n]$. Then*

(*i*) $\mathrm{Lk}_{\sigma_1 \cup \sigma_2}(\Delta) \subseteq \mathrm{Lk}_{\sigma_1}(\Delta) \cap \mathrm{Lk}_{\sigma_2}(\Delta)$, *and*

(*ii*) $\mathrm{Lk}_{\sigma_1}(\Delta) \cup \mathrm{Lk}_{\sigma_2}(\Delta) \subseteq \mathrm{Lk}_{\sigma_1 \cap \sigma_2}(\Delta)$.

Proof. For (i), suppose that $\tau \in \mathrm{Lk}_{\sigma_1 \cup \sigma_2}(\Delta)$. Then necessarily τ is disjoint from both σ_1 and σ_2, since it is disjoint from their union. Furthermore, $\tau \cup \sigma_1$ and $\tau \cup \sigma_2$ are both faces in Δ since $\tau \cup \sigma_1 \cup \sigma_2$ is a face of Δ and Δ is downclosed. Thus τ is in both $\mathrm{Lk}_{\sigma_1}(\Delta)$ and $\mathrm{Lk}_{\sigma_2}(\Delta)$.

To see that (ii) holds, let τ be in $\mathrm{Lk}_{\sigma_1}(\Delta) \cup \mathrm{Lk}_{\sigma_2}(\Delta)$. Since τ is disjoint from at least one of σ_1 and σ_2 it is disjoint from their intersection. If $\tau \in \mathrm{Lk}_{\sigma_1}(\Delta)$ then $\tau \cup \sigma_1$ is a face of Δ, and since Δ is downclosed, so must be $\tau \cup (\sigma_1 \cap \sigma_2)$. By symmetric reasoning, if $\tau \in \mathrm{Lk}_{\sigma_2}(\Delta)$ then $\tau \cup (\sigma_1 \cap \sigma_2)$ is again a face of Δ. Thus $\tau \in \mathrm{Lk}_{\sigma_1 \cap \sigma_2}(\Delta)$. □

1.4 Classes of Codes

The problem of determining whether an arbitrary neural code is convex has proven to be difficult, and remains an open problem Curto et al. (2015). As a result it is natural to consider codes which satisfy certain mathematical constraints. These restricted classes of codes can be easier to analyze for convexity, while still providing useful results. In this section we briefly mention several important classes of codes and provide an overview of the literature regarding them.

Definition 1.18 (Jeffs et al. (2015))**.** A code C is k-sparse if $|\sigma| \leq k$ for every $\sigma \in C$. Equivalently, if the dimension of $\Delta(C)$ is no larger than $k - 1$.

Biologically, a k-sparse code is one in which no more than k neurons fire simultaneously. Conveniently, most biological data is relatively sparse in nature. In Jeffs et al. (2015) the authors classified all 2-sparse codes which have convex realizations in $\mathbb{R}^3$.

Since simplicial complexes are well understood, one might hope for the support of a code to be a simplicial complex.

Definition 1.19 (Curto et al. (2013))**.** A code C is a *simplicial complex code* if $\mathrm{supp}(C)$ is a simplicial complex.

A slightly weaker condition than downclosure is that of being closed under intersection. Codes whose supports are closed under intersection are called intersection complete, and form a larger class of codes than simplicial complex codes.

Definition 1.20 (Curto et al. (2015))**.** A code C is intersection complete if it is closed under intersection. That is, if $\sigma \cap \tau \in C$ for all $\sigma, \tau \in C$.

As discussed in Curto et al. (2015), it is known that any intersection complete code is convex. Since simplicial complexes are closed under intersection this immediately implies that simplicial complex codes are also convex. Classifying codes by certain properties such as sparsity and intersection completeness is a useful avenue to understanding convex codes. Although the general problem of determining when a code is convex has proven difficult, these special cases help classify existing neural data concretely.

Chapter 2

An Algebraic Approach to Understanding Codes

In this chapter we approach codes from an algebraic perspective. By associating algebraic objects to codes we are able to understand their structure in new and revealing ways. We are also able to leverage traditional tools from fields such as algebraic geometry. We begin with an overview of some existing algebraic techniques in the study of neural codes, and discuss in particular the neural ideal of a code. In section 2.4 we consider algebraic maps between polynomial rings and their effects on codes, drawing a correspondence between these maps and "transformations" of codes. We will see in Chapter 3 that some of these transformations have natural geometric meanings.

2.1 Ideals and Varieties

In algebraic geometry two objects of concern are ideals and varieties. Varieties are zero sets of polynomials in a space. For example $\{(x, x^2) \mid x \in \mathbb{R}\}$ is a variety in $\mathbb{R}^2$ since it is the zero set of the polynomial $y - x^2$. Varieties are associated to the ideal in a polynomial ring consisting of all polynomials which vanish on the variety. This allows for translation between the worlds of algebra and geometry. To analyze codes we will consider the space $\mathbb{F}_2^n$ of all binary vectors on n bits. Our polynomials will come from the ring $\mathbb{F}_2[n] := \mathbb{F}_2[x_1, \ldots, x_n]$.

Definition 2.1 (Curto et al. (2013))**.** Let $C \subseteq \mathbb{F}_2^n$ be a code on n bits. The *ideal*

of C, denoted I_C, is the set

$$I_C := \{f \in \mathbb{F}_2[n] \mid f(c) = 0 \text{ for all } c \in C\}.$$

Proposition 2.2 (Curto et al. (2013))**.** *I_C is an ideal in $\mathbb{F}[n]$.*

Proof. Notice that I_C is never empty since it always contains the zero polynomial. Then let $f, g \in I_C$. Clearly $f - g \in I_C$ since for any $c \in C$ we have $(f - g)(c) = f(c) - g(c) = 0 - 0 = 0$. Similarly, $fg \in I_C$ since we will have $(fg)(c) = f(c)g(c) = 0 \cdot 0 = 0$. Thus I_C is a subring of $\mathbb{F}[n]$. Letting $h \in \mathbb{F}_2[n]$ we see that I_C is an ideal since $(hf)(c) = h(c)f(c) = h(c) \cdot 0 = 0$. □

Definition 2.3 (Curto et al. (2013))**.** Let $I \subseteq \mathbb{F}_2[n]$ be an ideal. Then the *variety* of I, denoted $V(I)$, is the set

$$V(I) := \{v \in \mathbb{F}_2^n \mid f(v) = 0 \text{ for all } f \in I\}.$$

Note that the variety of an ideal may be empty, for example in the case that $I = \mathbb{F}_2[n]$. For ideals I_C we have that $V(I_C) = C$, and ideals I_C and codes C are in 1-to-1 correspondence via this relationship. The I_C can be used to infer properties of C and its realizations, as the following example hints.

Example 2.4. Suppose that we are given an ideal $I_C \subseteq \mathbb{F}_2[3]$ which contains the polynomial $f = x_1(1 - x_2)$. Then C cannot contain either 100 or 101, since f does not vanish on these vectors.

There are some nonzero polynomials which are in I_C for every code C. In particular, the polynomial $x_i(1 - x_i)$ vanishes on every binary vector for any i. These polynomials are referred to as *Boolean*. Their presence in I_C is a product of the fact that we are working over a finite field, and they do not encode any useful information about our code. This motivates the content of the following section, which seeks to remedy the fact that I_C contains these trivial relations.

2.2 Neural Ideals

Instead of associating each code C to the ideal I_C, we seek to construct a *neural ideal* for a code, which captures only nontrivial information about the code. To do this we first examine pseudomonomials. We will see that pseudomonomials can be used to encode effectively all interesting information about a code.

Definition 2.5 (Curto et al. (2013))**.** A *pseudomonomial* is a polynomial $f \in \mathbb{F}_2[n]$ which is the product of independent linear factors in single variables. That is, we have that

$$f = \prod_{i \in \sigma} x_i \prod_{j \in \tau} (1 - x_j).$$

where $\sigma, \tau \subseteq [n]$ and $\sigma \cap \tau = \emptyset$.

In the definition above we adopt the usual convention that the empty product is 1. Notice that Boolean relations $x_i(1 - x_i)$ are *not* pseudomonomials since they have $\sigma = \tau = \{i\}$. For convenience we will write x_σ for $\prod_{i \in \sigma} x_i$ and $\overline{x}_\tau$ for $\prod_{j \in \tau}(1 - x_j)$ for the expressions in Definition 2.5, so that pseudomonomials are expressed as

$$f = x_\sigma \overline{x}_\tau.$$

Pseudomonomials have an important relationship to vectors in $\mathbb{F}_2^n$. In particular, we can uniquely associate each vector a degree n pseudomonomial which evaluates to zero on all other binary vectors.

Definition 2.6 (Curto et al. (2013))**.** Let $v \in \mathbb{F}_2^n$. Then the *indicator* for v is the pseudomonomial

$$\rho_v := \prod_{v_i = 1} x_i \prod_{v_j = 0} (1 - x_j).$$

Notice that $\rho_v(w) = 1$ if and only if $w = v$, and otherwise $\rho_v(w) = 0$. Also observe that every degree n pseudomonomial is an indicator polynomial for some vector: since we have n factors and cannot repeat variables we must have exactly one factor x_i or $\overline{x}_i$ for each i. It also worth noting that no pseudomonomial can have degree larger than n, again since each variable appears in at most one linear factor. Thus indicator polynomials are precisely the set of maximal pseudomonomials in $\mathbb{F}_2[n]$ with respect to division.

We are now prepared to establish a more precise translation between codes in $\mathbb{F}_2^n$ and ideals in $\mathbb{F}_2[n]$ by defining the *neural ideal* and *neural ring* of a code. Introduced in Curto et al. (2013), the neural ideal and ring are defined as follows.

Definition 2.7 (Curto et al. (2013))**.** Let $C \subseteq \mathbb{F}_2^n$ be a code on n bits. Then the *neural ideal* of C, denoted J_C, is the ideal

$$J_C := \langle \rho_v \mid v \notin C \rangle.$$

Here we adopt the convention that the ideal generated by the empty set is the zero ideal. Then *neural ring* of C is the quotient ring

$$R_C := \mathbb{F}_2[n]/I_C.$$

This ring can be thought of as the ring of functions $C \to \{0, 1\}$.

Notice that because J_C is generated by indicators for vectors not in C we have immediately that $J_C \subseteq I_C$. In fact, Curto et al. (2013) showed that the only difference between J_C and I_C is that I_C contains the Boolean relations while J_C need not. This allows us to decompose I_C neatly.

Definition 2.8 (Curto et al. (2013))**.** The *Boolean ideal* is the ideal $\mathcal{B} := \langle x_i \overline{x}_i \mid i \in [n] \rangle$ generated by all possible Boolean relations.

Lemma 2.9 (Curto et al. (2013))**.** *Let C be a code. Then $I_C = J_C + \mathcal{B}$.*

One can verify that $V(J_C) = C$. The following example provides some initial evidence that J_C is useful in determining the properties of codes and their realizations.

Example 2.10. Let $C = \{000, 100, 010, 110, 001\}$. The only codewords not in C are 111, 101, and 011. Thus the neural ideal of C is

$$J_C = \langle \rho_{111}, \rho_{101}, \rho_{011} \rangle = \langle x_1 x_2 x_3, x_1(1 - x_2)x_3, x(1 - x_1)x_2 x_3 \rangle.$$

However, there are more pseudomonomials in J_C than just the three indicators above. In particular, we can notice that $\rho_{111} + \rho_{101} = x_1 x_3$, and $\rho_{111} + \rho_{011} = x_2 x_3$. Because $x_1 x_3$ is in J_C, it must be the case that no codeword in C is of the form $1 * 1$ where $* \in \{0, 1\}$, since $x_1 x_3$ evaluates to 1 on such a codeword. But this means that in any realization of C we must have $U_1 \cap U_3 = \emptyset$. Similar logic tells us that since $x_2 x_3 \in J_C$, we must have $U_2 \cap U_3 = \emptyset$ in any realization of C. From this we see that the polynomials in J_C can encode important information about relationships between the sets in a realization of C.

In order to characterize the structure of neural ideals we return to pseudomonomials. Every neural ideal is generated by a collection of pseudomonomials (in particular, indicator polynomials) and so understanding how pseudomonomials behave is an important step towards understanding neural ideals. A natural first step in investigating pseudomonomials is to consider the ideals that they generate in $\mathbb{F}_2[n]$.

Definition 2.11. An ideal I in $\mathbb{F}_2[x_1, \ldots, x_n]$ is called a *pseudomonomial ideal* if it has a generating set consisting of pseudomonomials.

Our main result in this section will be the following:

Theorem 2.1. *An ideal $I \subseteq \mathbb{F}_2[n]$ is a neural ideal if and only if it is a pseudomonomial ideal.*

In order to arrive at this result we first develop some general tools related to pseudomonomials.

Lemma 2.12. *Let $f = x_\sigma \overline{x}_\tau$ be a pseudomonomial. Then $f(v) = 1$ if and only if f divides ρ_v.*

Proof. Suppose that $f(v) = 1$. To show that f divides ρ_v it suffices to show that each linear term of f is present in ρ_v. If a linear term in f is of the form x_i, then clearly $i \in \text{supp}(v)$ since otherwise f would vanish on v. Since $i \in \text{supp}(v)$ we know that x_i is a factor of ρ_v. This leaves linear terms of the form $(1 - x_j)$. In this case j cannot be in $\text{supp}(v)$, and so $(1 - x_j)$ is also a factor of ρ_v. Since each linear factor of f divides ρ_v and no factors are repeated, we conclude that f divides ρ_v. To show the converse suppose that f divides ρ_v. Clearly $f(v)$ cannot be zero since $\rho_v(v)$ is nonzero, and so we conclude that $f(v) = 1$. □

Definition 2.13. Let $f = x_\sigma \overline{x}_\tau$ be a pseudomonomial. Then the *canopy* of f is the collection of all indicator polynomials that f divides, denoted

$$\text{can}(f) := \{\rho_v \mid f \text{ divides } \rho_v\}.$$

Lemma 2.14. *Let $f = x_\sigma \overline{x}_\tau$ be a pseudomonomial. Then f is the sum of all polynomials in* $\text{can}(f)$*. That is, f is the sum of all indicator polynomials in $\mathbb{F}_2[n]$ that it divides.*

Proof. We will proceed by induction on $n - \deg(f)$. We observed earlier that pseudomonomials always have degree no larger than n, and so we begin with the base case that $n = \deg(f)$. In this case f is an indicator polynomial, and $\text{can}(f) = \{f\}$. Thus it is trivially the sum of all elements in its canopy.

Next let $k \geq 0$, and suppose that the lemma holds for pseudomonomials of degree $n - k$. Let f be a pseudomonomial of degree $n - k - 1$. Since f has degree strictly smaller than n, there must be some variable x_i that f does not depend on. Then $x_i f$ and $(1 - x_i)f$ are both pseudomonomials, which by the inductive hypothesis can be represented as the sum of all

elements in can$(x_i f)$ and can$((1 - x_i)f)$ respectively. Notice that can$(x_i f)$ and can$((1 - x_i)f)$ partition can(f), since every indicator that f divides has a factor of either x_i or $(1 - x_i)$ but not both. We then compute directly that

$$\begin{aligned} f &= x_i f + (1 - x_i) f \\ &= \left(\sum_{\rho_v \in \text{can}(x_i f)} \rho_v \right) + \left(\sum_{\rho_v \in \text{can}((1-x_i)f)} \rho_v \right) \\ &= \sum_{\rho_v \in \text{can}(f)} \rho_v. \end{aligned}$$

Thus f is the sum of all indicator polynomials that it divides. By induction, any pseudomonomial can be written as the sum of all elements of its canopy. □

Corollary 2.15. *The sum of all indicator polynomials in* $\mathbb{F}_2[n]$ *is* 1.

Corollary 2.16. *Every pseudomonomial in a neural ideal* J_C *can be written as the sum of indicators for vectors not in* C*.*

Proof. If f is a pseudomonomial in J_C then can$(f) \subseteq J_C$ since J_C is an ideal. Each indicator in the canopy of f thus must indicate a vector not in C. □

We can naturally extend the definition of a canopy to any ideal generated by pseudomonomials. Canopies of pseudomonomial ideals will be a significant tool in proving Theorem 2.1.

Definition 2.17. Let I be a pseudomonomial ideal. Then the *canopy* of I, denoted can(I), is the set of all indicator polynomials in I.

By Lemma 2.14, the canopy of a neural ideal will always form a generating set for the ideal. Furthermore, the canopy can be extracted from any generating set of pseudomonomials for a neural ideal.

Lemma 2.18. *Let* I *be a pseudomonomial ideal and let* $\{f_1, \ldots, f_k\}$ *be a generating set for* I *consisting of pseudomonomials. Then*

$$\text{can}(I) = \bigcup_{1 \leq i \leq k} \text{can}(f_i)$$

and can(I) *is a generating set for* I*.*

Proof. First let ρ_v be in the canopy of I. Then there must exist some f_i which does not vanish when evaluated on the codeword v, since ρ_v does not vanish on v. But then by Lemma 2.12 f_i divides ρ_v and so ρ_v is in $\text{can}(f_i)$. Conversely, any indicator polynomial that is a multiple of some f_i is necessarily in the ideal, and therefore in its canopy.

To see that $\text{can}(I)$ is a generating set, recall from Lemma 2.14 that each f_i is the sum of all polynomials in its canopy. The canopy of each f_i is contained in $\text{can}(I)$, and so clearly each f_i can be generated from $\text{can}(I)$. Thus $\langle \text{can}(I) \rangle$ contains I. The reverse inclusion is immediate since $\text{can}(I)$ is a subset of I. Therefore the canopy of a pseudomonomial ideal generates the ideal. □

Lemma 2.19. *Let I and J be pseudomonomial ideals. Then $I \subseteq J$ if and only if* $\text{can}(I) \subseteq \text{can}(J)$.

Proof. If $I \subseteq J$ then every indicator polynomial in I is an indicator polynomial in J, and so $\text{can}(I) \subseteq \text{can}(J)$. Conversely, since $\text{can}(I)$ and $\text{can}(J)$ are generating sets for I and J respectively we have that

$$I = \langle \text{can}(I) \rangle \subseteq \langle \text{can}(J) \rangle = J.$$

This proves the desired result. □

With these tools we now prove Theorem 2.1. In fact we show a slightly stronger result: each pseudomonomial ideal is the neural ideal of its variety.

Proof of Theorem 2.1. It is clear that neural ideals are pseudomonomial ideals since by Definition 2.7 the neural ideal is generated by indicator polynomials, which are themselves pseudomonomials. To prove the converse let $I = \langle f_1, \ldots, f_k \rangle$ be an ideal in $\mathbb{F}_2[n]$ with each f_i a pseudomonomial. Let $C = V(I)$ be the variety of I. We will show that $I = J_C$.

Since J_C and I are both pseudomonomial ideals we know from Lemma 2.19 that they are equal if and only if their canopies are equal. Let ρ_v be in the canopy of I. Clearly $v \notin C$ since all functions in I must vanish on all of C. But then ρ_v is in the canopy of J_C. To prove the reverse inclusion let ρ_u be in the canopy of J_C. Then $u \notin C = V(I)$. In particular, there must be some f_i for which $f_i(u) = 1$, since otherwise all functions in I would vanish at u and we would have $u \in C$. But by Lemma 2.12 we know that f_i divides ρ_u. In particular, ρ_u is in I and therefore in the canopy of I. This shows that the canopies of I and J_C are equal, and so $I = J_C$, proving the desired result. □

Henceforth we will not distinguish between neural ideals and pseudomonomial ideals. We conclude with a final observation regarding pseudomonomials which will prove useful in the following sections.

Lemma 2.20. *Let $f = x_\sigma \prod_{i\in\tau}(1 - x_i)$ be a pseudomonomial. Then f is the sum of all monomials x_γ with $\sigma \subseteq \gamma \subseteq \sigma \cup \tau$. In particular, f is a sum of exactly $2^{|\tau|}$ squarefree monomials.*

Proof. Since we are working over $\mathbb{F}_2$, we can replace the subtraction in each linear term of f with an addition. Doing so and expanding, we have that

$$\begin{aligned} f &= x_\sigma \prod_{i\in\tau}(1 + x_i) \\ &= x_\sigma \left(\sum_{\tau'\subseteq\tau} x_{\tau'} \right) \\ &= \sum_{\tau'\subseteq\tau} x_{\tau'\cup\sigma} \\ &= \sum_{\sigma\subseteq\gamma\subseteq\sigma\cup\tau} x_\gamma. \end{aligned}$$

Notice that there are $2^{|\tau|}$ ways to choose a set between σ and $\sigma \cup \tau$, and so this sum has exactly $2^{|\tau|}$ terms. This proves the desired result. □

2.3 The Canonical Form of a Neural Ideal

We saw in Example 2.10 that the presence of certain polynomials in the neural ideal J_C can yield information about C and its realizations. One of the most useful tools for extracting this information is the *canonical form* of a neural ideal.

Definition 2.21 (Curto et al. (2013))**.** Let C be a code and let J_C be its neural ideal. The *canonical form* of J_C, denoted $\mathrm{CF}(J_C)$, is the set of minimal pseudmonomials in J_C with respect to division. That is,

$$\mathrm{CF}(J_C) := \{f \in J_C \mid f \text{ is a pseudmonomial and no proper divisor of } f \text{ is in } J_C\}.$$

The canonical form will always be a generating set for J_C, since J_C is generated by indicators which are each necessarily the multiple of some pseudomonomial in $\mathrm{CF}(J_C)$. One might expect $\mathrm{CF}(J_C)$ to be a minimal generating set with respect to size, but this is not always the case, as the following example illustrates.

Example 2.22 (Curto et al. (2013))**.** Consider the pseudmonomial ideal $J_C = \langle x_1(1-x_2), x_2(1-x_3)\rangle \subset \mathbb{F}_2[3]$ (we know by Theorem 2.1 that this is the neural ideal for some code C.) Notice that this ideal cannot contain any proper divisors of $x_1(1-x_2)$ and $x_2(1-x_3)$, since these pseudmonomials do not generate such proper divisors. Thus $x_1(1-x_2)$ and $x_2(1-x_3)$ are both in $\mathrm{CF}(J_C)$. However, the canonical form contains an additional pseudomonomial, $x_1(1-x_3)$, since

$$\begin{aligned}(1-x_3)\big(x_1(1-x_2)\big) + x_1\big(x_2(1-x_3)\big) &= x_1(1-x_3)\big(1-x_2+x_2\big)\\ &= x_1(1-x_3).\end{aligned}$$

This is in fact the only other pseudomonomial in the canonical form, and so in this case $\mathrm{CF}(J_C) = \{x_1(1-x_2), x_2(1-x_3), x_1(1-x_3)\}$. Thus the canonical form need not be a minimal generating set in the usual sense of minimal.

2.4 Algebraic Transformations of Codes

There are many ways one might want to modify a code to obtain a new code. For example, one could restrict a code to a certain set of indices (cf. Definition 1.10), or take the link of a set in a code (cf. Definition 1.14), or permute the labeling of neurons. One way to operationalize the process of modifying codes is to consider a map $\mathcal{T}$ which takes a code as input and returns a modified code as output. In general, $\mathcal{T}$ can be thought of as an algorithm that modifies codes in some way. Given a code $\mathcal{C}$ and its canonical form $\mathrm{CF}(J_C)$, we want to find a way of describing $\mathrm{CF}(J_{\mathcal{T}(C)})$. Of course we can always compute $\mathrm{CF}(J_{\mathcal{T}(C)})$ directly from $\mathcal{T}(\mathcal{C})$, but we hope that if $\mathcal{T}$ is well-behaved then the canonical form of $\mathcal{T}(\mathcal{C})$ can be computed more quickly and easily from $\mathrm{CF}(J_C)$. Another potential task of interest is to characterize certain maps T which send convex codes to convex codes. In this section we present several classes of maps T which can be described in terms of ring homomorphisms $\mathbb{F}_2[n] \to \mathbb{F}_2[m]$. In Chapter 3 we will characterize certain maps that preserve convexity of codes.

2.4.1 Previous work

Some previous work has been done regarding transformations of codes and related effects on neural ideals. In Curto and Youngs (2015) the authors examine homomorphisms of neural rings. Recall that the neural ring of a code is the ring $R_C := R[n]/J_C$, consisting of functions $f : C \to \{0,1\}$. The

authors found that any function $q : \mathcal{C} \to \mathcal{D}$ between two codes naturally induced a ring homomorphism $q^* : R_{\mathcal{D}} \to R_{\mathcal{C}}$ defined by $q^*(f) = f \circ q$. To obtain more structure the authors considered neural rings as modules over $R[n]$, and characterized a limited class of homomorphisms between neural rings. These homomorphisms corresponded to functions between codes which were compositions of the following types of functions:

- Permutation of labels
- Adding a codeword (inclusion)
- Deleting the last neuron
- Duplicating a neuron
- Adding a trivial neuron

The authors also examined the canonical forms of codes and how they changed as a result of these maps. Several algorithms were provided for modifying the canonical form of an existing code to obtain the canonical form of the resulting code when one of these maps was applied.

2.4.2 Homomorphisms preserving neural ideals

In this section we consider ring homomorphisms $\phi : \mathbb{F}_2[n] \to \mathbb{F}_2[m]$ which map neural ideals to neural ideals. This differs from the work of Curto and Youngs (2015) since we are no longer considering maps between the neural rings themselves. We will see that these transformations have a natural meaning in terms of codes, and prove to be a useful tool in transforming canonical forms.

Definition 2.23. Let $\phi : \mathbb{F}_2[n] \to \mathbb{F}_2[m]$ be a homomorphism of rings. We say that ϕ *respects neural ideals* if the image of any neural ideal under ϕ is a neural ideal. That is, for every code $\mathcal{C} \subseteq \{0,1\}^n$ there is a code $\mathcal{D} \subseteq \{0,1\}^m$ so that

$$\phi(J_{\mathcal{C}}) = J_{\mathcal{D}}.$$

It is straightforward to observe that the identity map on $\mathbb{F}_2[n]$ respects neural ideals, and furthermore that compositions of maps that respect neural ideals will still respect neural ideals. This suggests that we can consider polynomial rings over $\mathbb{F}_2$ together with maps respecting neural ideals as a category. For the moment, we need not examine this perspective too closely,

but it will become relevant when we consider the effects of these maps on codes. For now, we focus on answering the following question:

Question 2.24. Which maps $\phi : \mathbb{F}_2[n] \to \mathbb{F}_2[m]$ respect neural ideals?

Since neural ideals are generated by pseudomonomials, it is natural to expect that any map respecting them should map pseudomonomials to pseudomonomials. It turns out that this is essentially the case, except that such a map may send some pseudomonomials to zero.

Lemma 2.25. *Let $\phi : \mathbb{F}_2[n] \to \mathbb{F}_2[m]$ be a homomorphism. Then ϕ respects neural ideals if and only if ϕ is surjective and the image of any pseudomonomial under f is either a pseudomonomial or zero.*

Proof. Recall from Theorem 2.1 that neural ideals are precisely pseudomonomial ideals. Thus it suffices to prove that ϕ respects pseudomonomial ideals if and only if the image of any pseudomonomial under f is either a pseudomonomial or zero.

($\Rightarrow$) First suppose that ϕ is not surjective. Notice that the pseudomonomial ideal $\langle 1 \rangle = \mathbb{F}_2[n]$ is not mapped to a pseudomonomial ideal. In particular, $\phi(\mathbb{F}_2[n])$ contains the unit $\phi(1) = 1$ but is not the whole ring $\mathbb{F}_2[m]$. Therefore $\phi(\mathbb{F}_2[n])$ is not an ideal. This shows that ϕ must be surjective.

Next let $f = x_\sigma \overline{x}_\tau$ be a pseudomonomial in $\mathbb{F}_2[n]$, and consider $\phi(f)$. If $\phi(f)$ is not a pseudomonomial then no multiple of $\phi(f)$ can be a pseudomonomial. Indeed, if $\phi(f)g = x_\sigma \overline{x}_\tau$ for some function $g \in \mathbb{F}_2[m]$ then we know that each x_i and $\overline{x}_j$ dividing the right hand side divide exactly one of $\phi(f)$ and g since x_i and $\overline{x}j$ are irreducible and $\mathbb{F}_2[m]$ is a unique factorization domain. As a result, if $\phi(f)$ is not a pseudmonomial then $\langle \phi(f) \rangle$ is an ideal of $\mathbb{F}_2[m]$ which contains no pseudomonomials. But the only pseudomonomial ideal containing no pseudomonomials is the zero ideal. Therefore $\phi(f) = 0$. Thus any homomorphism ϕ respecting neural ideals must map pseudomonomials to pseudomonomials or zero.

($\Leftarrow$) Let J_C be a neural ideal in $\mathbb{F}_2[n]$ with a generating set of pseudomonomials $\{f_1, \ldots, f_k\}$. Since ϕ is a surjective homomorphism $\phi(J_C)$ is an ideal in $\mathbb{F}_2[m]$. Furthermore, the set $\{\phi(f_1), \ldots, \phi(f_k)\} \setminus \{0\}$ is a generating set for $\phi(J_C)$ and consists of pseudomonomials since ϕ maps pseudomonomials to pseudomonomials or zero. We conclude that ϕ respects neural ideals. □

Corollary 2.26. *Let $\phi : \mathbb{F}_2[n] \to \mathbb{F}_2[m]$ be a homomorphism that respects neural ideals. Then $m \leq n$.*

Proof. Any homomorphism $\phi : \mathbb{F}_2[n] \to \mathbb{F}_2[m]$ is uniquely defined by its action on each variable x_i, since the set $\{x_i \mid i \in [n]\}$ is an algebraically independent set which generates $\mathbb{F}_2[n]$ as an algebra over $\mathbb{F}_2$. If ϕ is surjective then the image of this set must generate $\mathbb{F}_2[m]$, and in particular it must generate the set $\{x_i \mid i \in [m]\}$. But then we cannot have $m > n$ since there is no way for a set of n objects to generate a larger set of algebraically independent objects. □

Example 2.27. Let $m \leq n$ and consider the map $\phi : \mathbb{F}_2[n] \to \mathbb{F}_2[m]$ which sends $x_i \mapsto 0$ whenever $i > m$ and fixes all other variables. This is a homomorphism which respects neural ideals. One can use Lemma 2.25 and verify that the image of any pseudomonomial under this map is either a pseudomonomial or zero. This follows from the fact that each linear factor in a pseudomonomial is either fixed under this map, or is mapped to 0 or 1. The fact that ϕ is surjective is a result of the fact that $\phi(x_i) = x_i$ for all $i \leq m$.

Intuitively, the above map is getting rid of extra variables whenver $n > m$ by mapping them to zero. In fact we can generalize the above example by sending extra variables to either 0 or 1 as we choose. As we will see later, such maps correspond to restricting our code to a particular set of "consistent" codewords.

We now give some key examples of maps which respect neural ideals. The three types of maps described in what follows will serve as a basis for classifying all maps that respect neural ideals.

Definition 2.28. Let $1 \leq m \leq m' \leq n$ and define a map $\omega : \mathbb{F}_2[n] \to \mathbb{F}_2[m]$ so that

$$\omega(x_i) = \begin{cases} x_i & i \leq m \\ 0 & m < i \leq m' \\ 1 & m' < i \leq n \end{cases}$$

Such a map is called a *restriction map* from $\mathbb{F}_2[n]$ to $\mathbb{F}_2[m]$. When the parameters m and m' are not clear we will denote such a map $\omega_{m,m'}$.

Such a map divides the numbers 1 through n up onto three sections: the first m are fixed, the next $m' - m$ are sent to 0, and the remaining variables are sent to 1.

Lemma 2.29. *Restriction maps respect neural ideals.*

Proof. Let $m \leq m' \leq n$ and define $\omega : \mathbb{F}_2[n] \to \mathbb{F}_2[m]$ to be a restriction map as in Definition 2.28. The map ω is a homomorphism since it is an evaluation map between polynomial rings. Thus by Lemma 2.25 it suffices to show that ω is surjective and maps pseudomonomials to pseudomonomials or zero. Note that ω is surjective since it must map identity to identity and every variable $x_i \in \mathbb{F}_2[m]$ has a preimage under ω, namely itself.

To see that ω maps pseudomonomials to pseudomonomials or zero, let $f = x_\sigma \overline{x}_\tau$ be a pseudomonomial and notice that

$$\omega(f) = \prod_{i \in \sigma} \omega(x_i) \prod_{j \in \tau} (1 - \omega(x_j)).$$

Every term $\omega(x_i)$ is either 0, 1, or x_i while every term $(1 - \omega(x_j))$ is either $0, 1$, or $(1 - x_j)$. We therefore see that $\omega(f)$ is either zero or a product of independent linear factors and hence a pseudomonomial. Thus ω sends pseudomonomials to pseudomonomials or zero, and must respect neural ideals. □

Definition 2.30. For any $i \in [n]$ the *bit flipping map at index i* is the function $\delta_i : \mathbb{F}_2[n] \to \mathbb{F}_2[n]$ defined by

$$\delta_i(x_j) = \begin{cases} x_j & j \neq i \\ 1 - x_j & j = i. \end{cases}$$

extended algebraically to all of $\mathbb{F}_2[n]$. We will call any composition of these maps a *bit flipping map*.

Lemma 2.31. *Any bit flipping map is an involution and respects neural ideals.*

Proof. It suffices to show that any δ_i is an involution repsecting neural ideals since the indexed maps commute with one another. To see that δ_i is a homomorphism notice that we define it only on variables (and 1) and extend algebraically. The set of variables is an algebraically independent set which generates $\mathbb{F}_2[n]$ as an algebra and so extending δ_i from this set must yield a homomorphism.

To prove that δ_i respects neural ideals, first observe it affects only x_i and fixes all other variables. In particular, δ_i replaces x_i with $1 - x_i$. We can infer that δ_i maps pseudomonomials to pseudomonomials since any linear factor not depending on x_i remains fixed while linear factors depending on x_i "flip":

$$\begin{aligned} \delta_i(x_i) &= 1 - x_i, \\ \delta_i(1 - x_i) = \delta_i(1) - \delta_i(x_i) &= 1 - (1 - x_i) = x_i. \end{aligned}$$

The linear factors of any pseudomonomial therefore remain linear and independent under δ_i.

Finally, δ_i is surjective since the image of all variables under δ_i is the set

$$\{1 - x_i\} \cup \{x_j \mid j \in [n] \text{ and } j \neq i\}$$

which generates $\mathbb{F}_2[n]$ as an algebra. By Lemma 2.25 we conclude that δ_i is a homomorphism that respects neural ideals.

Since δ_i is a surjective homomorphism from $\mathbb{F}_2[n]$ to $\mathbb{F}_2[n]$ it must be an automorphism. It is also clear that δ_i is an involution since its action on any x_j twice yields x_j. When $j \neq i$ this is clear since δ_i fixes x_j. When $j = i$ we have that $\delta_i(x_i) = 1 - x_i$ and so

$$\delta_i(\delta_i(x_i)) = \delta_i(1 - x_i) = 1 - \delta(x_i) = 1 - (1 - x_i) = x_i.$$

Therefore δ_i is an involution as desired. □

Definition 2.32. Let λ be a permutation of $[n]$. Then the map which sends $x_i \mapsto x_{\lambda(i)}$ is called a *permutation map* of $\mathbb{F}_2[n]$.

Proposition 2.33. *Permutation maps respect neural ideals.*

The proposition above is relatively immediate, since permutation merely relabels our variables, and does nothing to change the structure of polynomials in $\mathbb{F}_2[n]$. Thus a permutation map must send pseudomonomials to pseudomonomials, and must be bijective since permutations of $[n]$ are necessarily bijective.

So far we have constructed three types of maps that respect neural ideals: restrictions, bit flipping maps, and permutations. The remainder of this section will be dedicated to proving that, up to composition, these are the *only* maps that respect neural ideals.

Theorem 2.2. *Let $\phi : \mathbb{F}_2[n] \to \mathbb{F}_2[m]$ be a map respecting neural ideals. Then ϕ can be expressed as a composition of the following types of maps*:

- *Restriction* (*Definition* 2.28),
- *Bit flipping* (*Definition* 2.30), *and*
- *Permutation* (*Definition* 2.32).

In particular, we can write $\phi = \omega \circ \delta \circ \lambda$ where ω is a restriction, δ is a bit flipping map, and λ is a permutation.

To arrive at this result we first establish several lemmas.

Lemma 2.34. *Let $\phi : \mathbb{F}_2[n] \to \mathbb{F}_2[m]$ be a map respecting neural ideals. Then $\phi(x_i)$ is either a constant or a linear polynomial. In particular, $\phi(x_i) \in \{0, 1, x_j, 1 - x_j\}$.*

Proof. First notice that the set

$$\{1\} \cup \{x_i \mid i \in [n]\}$$

generates $\mathbb{F}_2[n]$ as an algebra. That is, every element of $\mathbb{F}_2[n]$ can be written as an algebraic combination of elements of this set. Since ϕ is surjective by Lemma 2.25 and $\phi(1) = 1$ it must be the case that

$$\{1\} \cup \{\phi(x_i) \mid i \in [n]\}$$

generates $\mathbb{F}_2[m]$ as an algebra. In particular, this set must contain at least m nonconstant polynomials. Then the collection of indices

$$\alpha = \{i \in [n] \mid \phi(x_i) \text{ is not a constant } \}$$

has at least m elements. Since ϕ respects neural ideals we also know from Lemma 2.25 that the image of the (pseudo)monomial x_α will also be a pseudomonomial. Now notice that

$$\phi(x_\alpha) = \phi\left(\prod_{i \in \alpha} x_i\right) = \prod_{i \in \alpha} \phi(x_i).$$

Since none of the $\phi(x_i)$ are constant, this pseudomonomial has degree at least $|\alpha| \geq m$. Since $\phi(x_\alpha)$ is a pseudomonomial in $\mathbb{F}_2[m]$ it cannot have degree larger than m and we conclude that each $\phi(x_i)$ must be linear. Therefore $\phi(x_i)$ is either a constant or linear for all $i \in [n]$. The only constants in $\mathbb{F}_2[m]$ are 0 and 1, and the only linear pseudomonomials are of the form x_j and $1 - x_j$. Therefore $\phi(x_i) \in \{0, 1, x_j, 1 - x_j\}$ for each i as desired. □

We can strengthen this result slightly by showing that each linear factor in $\mathbb{F}_2[m]$ has a *unique* preimage under any homomorphism that respects neural ideals.

Lemma 2.35. *Let $\phi : \mathbb{F}_2[n] \to \mathbb{F}_2[m]$ be a map respecting neural ideals. Then for each $j \in [m]$ there exists a unique $i \in [n]$ so that $\phi(x_i) \in \{x_j, 1 - x_j\}$.*

Proof. We showed in Lemma 2.34 that if $\phi(x_i)$ is not a constant then it must be either x_j or $1 - x_j$ for some $j \in [m]$. Since ϕ respects neural ideals it is surjective by Lemma 2.25, and so there must exist some i so that $\phi(x_i)$ depends on x_j. It remains to show that such an i is unique.

Suppose for contradiction that for $i \neq k$ we have $\phi(x_i)$ and $\phi(x_k)$ are both in $\{x_j, \overline{x}_j\}$. Then notice that since $i \neq k$ the monomial $x_i x_k$ is a pseudomonomial in $\mathbb{F}_2[n]$. But then

$$\phi(x_i x_k) = \phi(x_i)\phi(x_k)$$

is a product of dependent linear factors in $\mathbb{F}_2[m]$, and therefore not a pseudomonomial. Thus ϕ does not respect neural ideals by Lemma 2.25, a contradiction. Therefore for every $j \in [m]$ the map ϕ must send precisely one variable x_i to a linear pseudomonomial depending on x_j. □

With these tools established we can return to our main result. The following proof is constructive, in that it provides an explicit process for decomposing any map that respects neural ideals into a composition of a permutation, bit flipping maps, and restrictions.

Proof of Theorem 2.2. Let $\phi : \mathbb{F}_2[n] \to \mathbb{F}_2[m]$ be a homomorphism that respects neural ideals. We will decompose ϕ directly as follows. First partition $[n]$ into four sets:

$$\begin{aligned}
\alpha_0 &= \{i \in [n] \mid \phi(x_i) = x_j\} \\
\alpha_1 &= \{i \in [n] \mid \phi(x_i) = 1 - x_j\} \\
\beta_0 &= \{i \in [n] \mid \phi(x_i) = 0\}, \\
\beta_1 &= \{i \in [n] \mid \phi(x_i) = 1\}.
\end{aligned}$$

Recall from Corollary 2.26 that we must have $m \leq n$, and define a permutation λ of $[n]$ so that

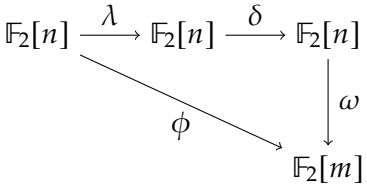
$$\lambda(i) = \begin{cases} j & \text{such that } \phi(x_i) \in \{x_j, 1 - x_j\} \text{ whenever } i \in \alpha_0 \cup \alpha_1 \\ m + j & \text{if } i \text{ is the } j\text{-th number in } \beta_0 \\ m + |\beta_0| + j & \text{if } i \text{ is the } j\text{-th number in } \beta_1 \end{cases}$$

This is indeed a permutation since by Lemma 2.35 each $i \in \alpha_0 \cup \alpha_1$ is uniquely associated to some $j \in [m]$, and the behaviour of λ on the remaining indices is necessarily bijective since no two indices can both be in the j-th position in β_0 or β_1. Then define δ to be the composition of all bit

flipping maps δ_i for all $i \in \lambda(\alpha_1)$. Finally, define $\omega : \mathbb{F}_2[n] \to \mathbb{F}_2[m]$ to be a restriction map so that

$$\omega(x_i) = \begin{cases} x_i & i \leq m \\ 0 & m < i \leq n + |\beta_0| \text{ (That is, whenever } i = \lambda(j) \text{ for some } j \in \beta_0) \\ 1 & m + |\beta_0| < i \text{ (That is, whenever } i = \lambda(j) \text{ for some } j \in \beta_1). \end{cases}$$

We now claim that $\phi = \omega \circ \delta \circ \lambda$, or equivalently that the following diagram commutes:



To prove this it suffices to show that the image of any x_i under $\omega \circ \delta \circ \lambda$ is the same as under ϕ. We consider the four cases which arise from our original partition of $[n]$.

Case 1: $i \in \alpha_0$ In this case δ will not affect $\lambda(x_i)$ since $\lambda(i) \notin \lambda(\alpha_1)$. Thus we need only consider $\omega(\lambda(x_i))$. Since $i \in \alpha_0$ we know that $\lambda(x_i) = x_j$ for some $j \leq m$, and from the definition of ω we see that $\omega(\lambda(x_i)) = \omega(x_j) = x_j$. From the definition of λ we know that $\phi(x_i) = x_j$, and so our composition of maps is consistent with ϕ in this case.

Case 2: $i \in \alpha_1$ Here $\lambda(i) \in \lambda(\alpha_1)$, and so δ will flip our variable. In particular, $\omega(\delta(\lambda(x_i))) = \omega(\delta(x_{\lambda(i)})) = \omega(1 - x_{\lambda(i)}) = 1 - \omega(\lambda(x_i))$. A similar argument to the previous case tells us that, by construction, $1 - \omega(\lambda(x_i)) = 1 - x_j = \phi(x_i)$. Thus $\phi(x_i) = \omega(\delta(\lambda(x_i)))$ in this case as well, as desired.

Case 3: $i \in \beta_0$ As in the first case δ will have no effect. The map λ will send x_i to x_j for some $m < j \leq m + |\beta_0|$, and so $\omega(\delta(\lambda(x_i))) = \omega(\lambda(x_i)) = \omega(x_j) = 0 = \phi(x_i)$.

Case 4: $i \in \beta_1$ As in the previous case δ will not flip $x_{\lambda(i)}$. However, λ will now map x_i to x_j where $m + |\beta_0| < j$. Then by the definition of ω we see that $\omega(\delta(\lambda(x_i))) = \omega(\lambda(x_i)) = \omega(x_j) = 1 = \phi(x_i)$.

We conclude that the map $\omega \circ \delta \circ \lambda$ is consistent with ϕ on all x_i. Since both are homomorphisms and the set of variables $\{x_i \mid i \in [n]\}$ generates $\mathbb{F}_2[n]$ as an algebra, we conclude that these maps are equal. □

Corollary 2.36. *Suppose that $\phi : \mathbb{F}_2[n] \to \mathbb{F}_2[m]$ is a map respecting neural ideals. Then for every pseudomonomial $f \in \mathbb{F}_2[m]$ there exists a pseudomonomial $\hat{f} \in \mathbb{F}_2[n]$ so that $\phi(\hat{f}) = f$.*

Proof. By the Theorem 2.2 it suffices to show that f has a pseudomonomial in its preimage under each of the three types of maps. For bit flipping and permutation this is clear since they have inverses which respect neural ideals, and hence map f to some pseudomonomial. For the case of restriction, we can let $\hat{f} = f$. □

Corollary 2.37. *Suppose that $\phi : \mathbb{F}_2[n] \to \mathbb{F}_2[m]$ is a map respecting neural ideals. Then for every neural ideal $J_{\mathcal{D}}$ in $\mathbb{F}_2[m]$ there exists a neural ideal J_C in $\mathbb{F}_2[n]$ so that $\phi(J_C) = J_{\mathcal{D}}$.*

Proof. For every indicator ρ_v in the canopy of $J_{\mathcal{D}}$ we can choose some pseudomonomial $\hat{\rho_v}$ in $\mathbb{F}_2[n]$ whose image is ρ_v. This can be done because the preimage of any pseudmonomial under permutation and bit flipping maps is a pseudomonomial, and restriction maps from $\mathbb{F}_2[n]$ to $\mathbb{F}_2[m]$ fix pseudomonomials in $\mathbb{F}_2[m]$. The ideal generated by all $\hat{\rho_v}$ is a neural ideal in $\mathbb{F}_2[n]$ whose image under ϕ is precisely $J_{\mathcal{D}}$. □

It is worth seeing an example of the decomposition described in Theorem 2.2. The given proof makes the maps δ, λ and ω very explicit, but not entirely concrete. One way to view this decomposition is by considering their action on the variables in $\mathbb{F}_2[n]$ visually.

Example 2.38. Consider the map $\phi : \mathbb{F}_2[6] \to \mathbb{F}_2[3]$ defined by

$$\begin{aligned} x_1 &\mapsto 0 & x_4 &\mapsto \overline{x}_1 \\ x_2 &\mapsto \overline{x}_3 & x_5 &\mapsto 1 \\ x_3 &\mapsto 1 & x_6 &\mapsto x_2 \end{aligned}$$

We can draw this map and its decomposition by considering the action of ϕ on variables. This illustration is provided in Figure 2.1, where black lines indicate quantities which map according to the labels (for example $x_1 \mapsto x_4$ or $x_6 \mapsto 1$) while red indicates mappings of the form $x_i \mapsto \overline{x}_j$.

It is worth noting that the three types of maps we have described do not commute with one another. Permutations, for example, change the labelling on variables and restriction and bit-flipping depend heavily on

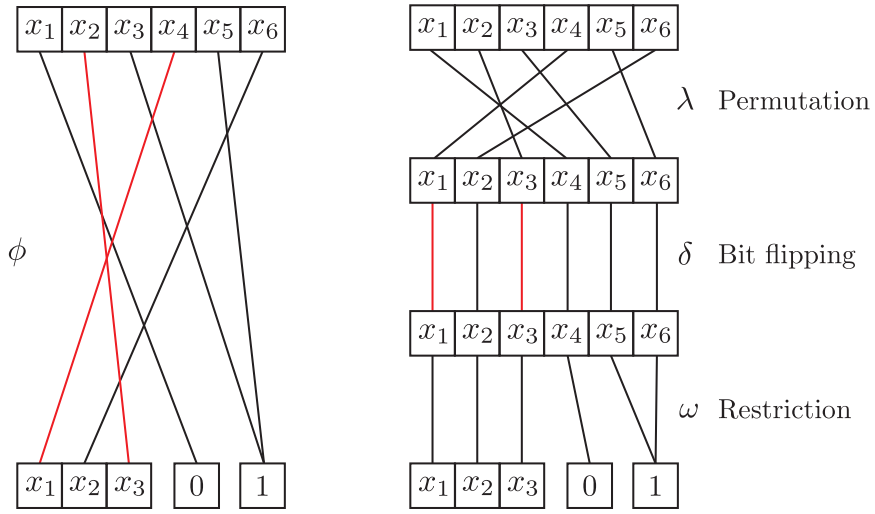


Figure 2.1 The homomorphism ϕ written as the composition of permutation, bit flipping, and restriction maps. Red lines indicate places where $x_i \mapsto \overline{x}_j$.

these variables. Thus it is relatively straightforward to see that permutation does not commute with bit flipping or restriction. Of course bit flipping cannot commute with restriction whenever the variable being "flipped" is eliminated by the restriction, since after applying the restriction map we can no longer flip the eliminated bit. On the other hand, restriction fixes all other variables, and so flipping a variable that is not eliminated does commute with restriction. These features indicate that we cannot decompose a general homomorphism in any way we please, though the decomposition provided in Theorem 2.2 is far from unique.

Having characterized all homomorphisms $\phi : \mathbb{F}_2[n] \to \mathbb{F}_2[m]$ which respect neural ideals, the question still remains: how do these homomorphisms affect codes? In particular, if $\phi(J_C) = J_{\mathcal{D}}$, then what can we say about the code $\mathcal{D}$ based on the code C? We will provide a satisfactory answer to this question in Section 2.4.4, but first we broaden our perspective slightly. So far we have considered only maps between the ambient polynomial rings $\mathbb{F}_2[n]$. It turns out that most of the information we care about in neural ideals is preserved if instead we consider them in rings of the form $\mathbb{F}_2[n]/\mathcal{B}$. Working in such rings can allow for greater freedom, and allows us to broaden the class of homomorphisms that respect neural ideals. The

following section deals in detail with this approach.

2.4.3 Homomorphisms modulo the Boolean relations

Recall that we are using the notation of Curto and Youngs (2015), in which $R[n] = \mathbb{F}_2[n]/\mathcal{B}$. To begin, we show that we can think of neural ideals as living in $R[n]$ instead of $\mathbb{F}_2[n]$ without any loss of generality. In particular, pseudomonomials are unaffected by projection modulo $\mathcal{B}$, and so the image of a neural ideal under projection mod $\mathcal{B}$ still encodes the full combinatorial information of a code.

When we speak of a neural ideal in $R[n]$ we are generalizing Definition 2.7, and considering these ideals as generated by a set of indicator pseudomonomials. Notice that indicator polynomials (in fact any pseudomonomials) are nonzero mod $\mathcal{B}$ since they contain no factors of the form $x_i\overline{x}_i$. From this we have that neural ideals in $R[n]$ are always the projection mod $\mathcal{B}$ of a neural ideal in $\mathbb{F}_2[n]$. Our first order of business is to observe that this projection is bijective.

Proposition 2.39. *Let $\pi_{\mathcal{B}} : \mathbb{F}_2[n] \to R[n]$ be projection mod $\mathcal{B}$. Then every neural ideal in $R[n]$ is of the form $\pi_{\mathcal{B}}(J_C)$ for some unique neural ideal $J_C \subseteq \mathbb{F}_2[n]$.*

This follows immediately from the observation that $\pi_{\mathcal{B}}$ maps pseudomonomials in $\mathbb{F}_2[n]$ to pseudomonomials in $R[n]$ bijectively. Henceforth we will write J_C for neural ideals in $R[n]$, with projection mod $\mathcal{B}$ implicit. As in the previous section, we say that a map $\phi : R[n] \to R[m]$ *respects neural ideals* if the image of any neural ideal under ϕ is again a neural ideal. We can pose the question from the previous section in this new context.

Question 2.40. Which maps $\phi : R[n] \to R[m]$ respect neural ideals?

It is not immediately obvious that the answer will be any different than in the previous section. Indeed, our starting point will be a lemma directly analogous to Lemma 2.25.

Lemma 2.41. *Let $\phi : R[n] \to R[m]$ be a homomorphism. Then ϕ respects neural ideals if and only if ϕ is surjective and the image of any pseudomonomial under f is either a pseudomonomial or zero.*

Proof. Observe that Theorem 2.1 still applies to neural ideals in $R[n]$, precisely because pseudomonomials are unaffected by projection mod $\mathcal{B}$. Then the arguments given in the proof of Lemma 2.25 can be used word for word to prove this lemma. The only small subtlety is that x_i and $\overline{x}_i$ are still nonzero and irreducible in $R[n]$, and $R[n]$ is a unique factorization domain. □

Recall in our proof of Lemma 2.35 that we used the fact that products of dependent factors could not be pseudomonomials in $\mathbb{F}_2[n]$. When working mod $\mathcal{B}$ we no longer have this restriction. In fact, we have by definition that $x_i(1-x_i)=0$ when we take the quotient of $\mathbb{F}_2[n]$ by $\mathcal{B}$, which implies $x_i^2 = x_i$. This also allows us to write

$$(1-x_i)^2 = 1 - 2x_i + x_i^2 = 1 + x_i = 1 - x_i.$$

So, while x_i^2 and $(1-x_i)^2$ were not pseudomonomials in $\mathbb{F}_2[n]$, they reduce to pseudomonomials when we project mod $\mathcal{B}$. In particular, we have just proven that $x_i^2 = x_i$ and $\overline{x}_i^2 = \overline{x}_i$ in $R[n]$. This difference in behavior allows us to prove the following lemma, which will serve as a basis for completely characterizing homomorphisms between rings $R[n]$ and $R[m]$ which respect neural ideals. This lemma gives us a more concrete characterization than that of Lemma 2.41.

Lemma 2.42. *Let $\phi : R[n] \to R[m]$ be a homomorphism. Then ϕ respects neural ideals if and only if ϕ is surjective and $\phi(x_i) \in \{0, 1, x_j, 1-x_j\}$ for all $i \in [n]$.*

Proof. ($\Rightarrow$) We know from Lemma 2.41 that ϕ must map pseudomonomials to pseudomonomials or 0. In particular, we know that $\phi(x_i)$ and $\phi(1-x_i) = 1 - \phi(x_i)$ are both pseudomonomials or 0. If $\phi(x_i)$ is zero, then it is clearly in $\{0, 1, x_j, 1-x_j\}$. If $1-\phi(x_i)$ is zero then $\phi(x_i) = 1$, and again $\phi(x_i)$ is of the desired form.

This leaves the case that $\phi(x_i)$ and $1-\phi(x_i)$ are both nonzero. In this case we know they are both pseudomonomials, and so by Lemma 2.20 there exist integers k and k' so that $\phi(x_i)$ is the sum of 2^k squarefree monomials and $1-\phi(x_i)$ is the sum of $2^{k'}$ squarefree monomials. But then $2^k = 2^{k'} \pm 1$, depending on whether 1 is one of the pseudomonomials in the sum defining $\phi(x_i)$. The only powers of 2 that differ by 1 are 1 and 2. Thus $\phi(x_i)$ is the sum of a single monomial or two monomials. The only pseudomonomials satisfying this are $\{1, x_j, 1-x_j\}$. This proves that $\phi(x_i) \in \{0, 1, x_j, 1-x_j\}$ for any variable x_i. Finally, it is automatic from Lemma 2.41 that ϕ must also be surjective.

($\Leftarrow$) By Lemma 2.41 it suffices to show that ϕ sends pseudomonomials to pseudomonomials or zero, since ϕ is surjective by hypothesis. To this end, notice that under ϕ the linear factors making up any pseudomonomial are mapped either to linear factors, zero, or one. In particular, we have that

$\phi(x_i) \in \{0, 1, x_j, 1 - x_j\}$ and

$$\begin{aligned}\phi(1 - x_i) &= 1 - \phi(x_i) \\ &\in \{1 - 0, 1 - 1, 1 - x_j, 1 - (1 - x_j)\} \\ &= \{0, 1, x_j, 1 - x_j\}.\end{aligned}$$

Now consider the image of any pseudomonomial under ϕ. Each linear term remains linear, but they may be dependent on one another. We have seen that $x_i^2 = x_i$ and $\overline{x}_i^2 = \overline{x}_i$ in $R[n]$, and so repeated linear factors do not cause any issues. If we obtain $x_i\overline{x}_i$ as a factor however, the pseudomonomial vanishes under ϕ since $x_i(1-x_i) = 0$ in $R[n]$. Thus every pseudomonomial is mapped to either another pseudomonomial or 0 under ϕ, and we conclude the desired result by Lemma 2.41. □

As a result of this lemma we can observe that all the maps respecting neural ideals that were described in the previous section also respect neural ideals when translated to $R[n]$. We defined these maps by their action on the set of variables x_i, and these definitions naturally apply in rings $R[n]$ since the set of variables is still an algebraically independent set that generates all of $R[n]$.

Corollary 2.43. *All of the following types of maps from $R[n]$ to $R[m]$ are homomorphisms that respect neural ideals*:

- *Restriction* (*Definition* 2.28)
- *Bit flipping* (*Definition* 2.30)
- *Permutation* (*Definition* 2.32).

Proof. Each of these maps is surjective by construction, and $\phi(x_i) \in \{0, 1, x_j, 1-x_j\}$ for each type of map as well. □

Corollary 2.44. *Any map $\phi : \mathbb{F}_2[n] \to \mathbb{F}_2[m]$ respecting neural ideals naturally induces a map $\phi' : R[n] \to R[m]$ respecting neural ideals.*

Notice, however, that in Lemma 2.42 we no longer require there to be a unique i for which $\phi(x_i) \in \{x_j, 1 - x_j\}$ for all j. This will allow for homomorphisms respecting neural ideals where there are "collisions" between variables. For example, we could let $\phi(x_1) = \phi(x_2) = x_1$, or let $\phi(x_1) = x_1$ and $\phi(x_2) = 1 - x_1$. This extra degree of freedom is precisely captured by the notion of an "identification" of variables.

Definition 2.45. Let $\eta : [n] \to [m]$ be a surjective function. Then the *identification map* induced by η is a map from $R[n]$ to $R[m]$ for which $x_i \mapsto x_{\eta(i)}$. Similar to permutations, we will abuse notation and write identification maps as $\eta : R[n] \to R[m]$ since the action of η on $R[n]$ is unambiguous.

Remark 2.46. Any permutation is an identification map, since a permutation is a surjective function from $[n]$ to $[n]$.

Notice that identification maps are homomorphisms that respect neural ideals by Lemma 2.41. Also notice that our only allowed identifications are those of the form $\phi(x_i) = \phi(x_j) = x_k$ for $i \neq j$. To obtain identifications of the form $\phi(x_i) = x_k$ and $\phi(x_j) = 1 - x_k$ or $\phi(x_i) = \phi(x_j) = 1 - x_k$ we need only compose identification maps with bit flips. With these maps we can now prove the following analog of Theorem 2.2.

Theorem 2.3. *Let $\phi : R[n] \to R[m]$ be a map respecting neural ideals. Then ϕ can be expressed as $\phi = \eta \circ \phi'$ where η is an indentification map and ϕ' is a map induced by some homomorphism between $\mathbb{F}_2[n]$ and $\mathbb{F}_2[m]$ respecting neural ideals.*

Proof. Let $\phi : R[n] \to R[m]$ be a map respecting neural ideals. Similar to the proof of Theorem 2.2 we will partition $[n]$ into four sets:

$$\begin{aligned}
\alpha_0 &= \{i \in [n] \mid \phi(x_i) = x_j\} \\
\alpha_1 &= \{i \in [n] \mid \phi(x_i) = 1 - x_j\} \\
\beta_0 &= \{i \in [n] \mid \phi(x_i) = 0\}, \\
\beta_1 &= \{i \in [n] \mid \phi(x_i) = 1\}.
\end{aligned}$$

Let $\alpha = \alpha_0 \cup \alpha_1$ be the set of indices whose variables are mapped to non-constant polynomials, and define $n' = |\alpha|$. Let λ be a permutation of $[n]$ so that $\lambda(\alpha) = [n']$ (i.e., λ puts the indices in α before all others). Now define a map $\phi' : R[n] \to R[n']$ so that

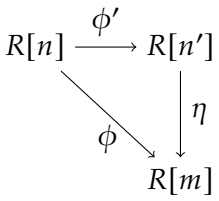
$$\phi'(x_i) = \begin{cases} x_{\lambda(i)} & i \in \alpha_0 \\ 1 - x_{\lambda(i)} & i \in \alpha_1 \\ 0 & i \in \beta_0 \\ 1 & i \in \beta_1. \end{cases}$$

Notice that for every index in $j \in [n']$ there is a unique x_i so that $\phi'(x_i) \in \{x_j, 1 - x_j\}$. Furthermore ϕ' maps each variable to either a linear pseudomonomial or a constant. Thus ϕ' is the induced map of some homomorphism $\mathbb{F}_2[n] \to \mathbb{F}_2[n']$ respecting neural ideals. To deal with possible

identification behavior of our map ϕ we define an indentification map $\eta : R[n'] \to R[m]$ as follows:

$$\eta(x_{\lambda(i)}) = x_j \text{ where } \phi(x_i) \in \{x_j, 1 - x_j\}.$$

We then claim that $\phi = \eta \circ \phi'$, or equivalently that the following diagram commutes:



If we can show that $\phi = \eta \circ \phi'$ then we are done, since η is an identification map and ϕ' is the induced map of a homomorphism $\mathbb{F}_2[n] \to \mathbb{F}_2[n']$ respecting neural ideals. As in the proof of Theorem 2.2, it suffices to show that ϕ and $\eta \circ \phi'$ agree on each variable. We consider two cases.

Case 1: $i \in \alpha_0$ or ß $\in \alpha_1$: Let j be the index so that $\phi(x_i) \in \{x_j, 1 - x_j\}$. If $i \in \alpha_0$ then we can compute that $\eta(\phi'(x_i)) = \eta(x_{\lambda(i)}) = x_j = \phi(x_i)$. Similarly, if $i \in \alpha_1$ we have that $\eta(\phi'(x_i)) = \eta(1 - x_{\lambda(i)}) = 1 - \eta(x_{\lambda(i)}) = 1 - x_j = \phi(x_i)$. In either case we see that ϕ and $\eta \circ \phi'$ agree.

Case 2: $i \in \beta_0$ or $i \in \beta_1$ In either of these cases that $\phi'(x_i)$ is constant, and so we have that $\eta(\phi'(x_i)) = \phi'(x_i) = \phi(x_i)$ since ϕ' and ϕ agree on variables that get mapped to constants. Thus the maps ϕ and $\eta \circ \phi'$ agree in this case as well. We conclude that $\phi = \eta \circ \phi'$ as desired. □

Corollary 2.47. *Suppose that $\phi : R[n] \to R[m]$ is a map respecting neural ideals. Then $\phi = \eta \circ \omega \circ \delta \circ \lambda$ where η is an indentification map, ω is a restriction map, δ is a bit flipping map, and λ is a permutation.*

Corollary 2.48. *Suppose that $\phi : R[n] \to R[m]$ is a map respecting neural ideals. Then for every pseudomonomial $f \in R[m]$ there exists a pseudomonomial $\hat{f} \in R[n]$ so that $\phi(\hat{f}) = f$.*

Proof. As in the proof of Corollary 2.36 it suffices to show that f is mapped to by some pseudomonomial under indentification maps, permutations, bit flipping maps, and restrictions. The latter three were discussed in Corollary 2.36 and so we consider only identification maps. Since f is a pseudomonomial we can write it as $f = x_\sigma \overline{x}_\tau$ for disjoint sets σ and τ. But then we can just let $\hat{f} = x_{\eta^{-1}(\sigma)} \overline{x}_{\eta^{-1}(\tau)}$. This is indeed a pseudomonomial since

disjointness of σ and τ implies disjointness of their preimages under η. It is clear that $\eta(\hat{(}f)) = f$ in this case since $\eta(\eta^{-1}(\sigma)) = \sigma$ and $\eta(\eta^{-1}(\tau)) = \tau$. □

Corollary 2.49. *Suppose that $\phi : R[n] \to R[m]$ is a map respecting neural ideals. Then for every neural ideal $J_{\mathcal{D}}$ in $R[m]$ there exists a neural ideal $J_{\mathcal{C}}$ in $R[n]$ so that $\phi(J_{\mathcal{C}}) = J_{\mathcal{D}}$.*

Proof. This is exactly the same as the proof of Corollary 2.37 with $\mathbb{F}_2[n]$ and $\mathbb{F}_2[m]$ replaced by $R[n]$ and $R[m]$. □

Theorem 2.3 shows us that we obtain slightly more freedom when working in $R[n]$. We have established in Proposition 2.39 that working in $R[n]$ preserves all relevant information contained in neural ideals, and so this extra freedom comes at no extra cost or loss of generality. In fact, computations in $R[n]$ are generally much easier since $R[n]$ is finite. The following sections make use of this fact, and return to the question of how these maps correspond to modifications of codes as described at the beginning of the chapter.

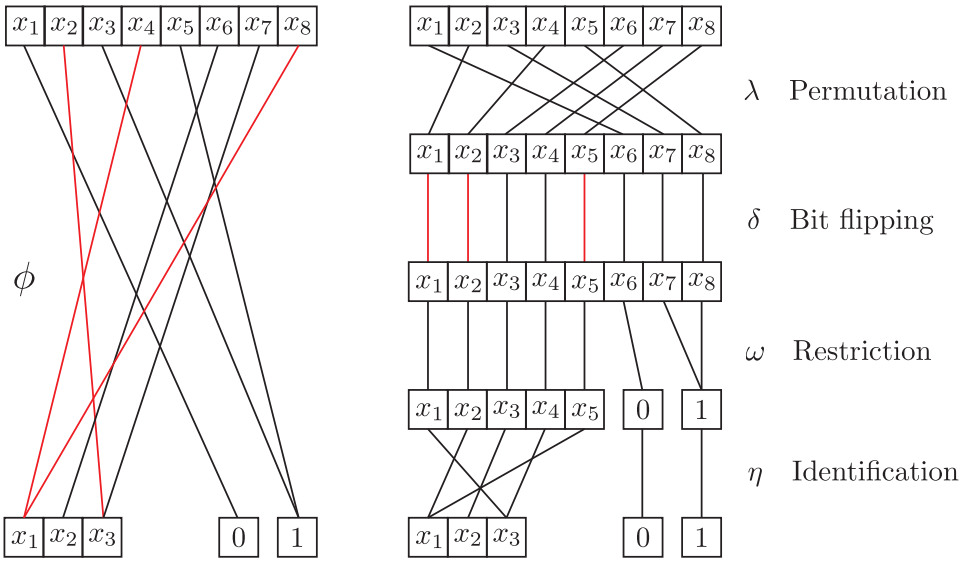
We conclude with an example of decomposing one of these maps.

Example 2.50. Let $\phi : R[8] \to R[3]$ be a homomorphism that respects neural ideals defined so that

$$\begin{aligned} x_1 &\mapsto 0 & x_5 &\mapsto 1 \\ x_2 &\mapsto \overline{x}_3 & x_6 &\mapsto x_2 \\ x_3 &\mapsto 1 & x_7 &\mapsto x_3 \\ x_4 &\mapsto \overline{x}_1 & x_8 &\mapsto \overline{x}_1 \end{aligned}$$

Notice that this is effectively an extension of the map defined in Example 2.1 which now includes identification behavior. We will write out its full decomposition into a permutation, bit flipping map, restriction, and identification map. This is illustrated below. As in Example 2.1 black lines indicate quantities which map according to the labels in the figure, while

red indicates mapping x_i to $\overline{x}_j$.



2.4.4 Induced Code Transformations

Recall from Theorem 2.3 that every homomorphism $\phi : R[n] \to R[m]$ respecting neural ideals can be expressed as the composition of identification maps, bit flipping maps, restrictions, and permutations. We now seek to answer the question of how these ring homomorphisms change the underlying codes. In particular we want to answer the following question.

Question 2.51. If $\phi(J_C) = J_{\mathcal{D}}$ for some homomorphism ϕ respecting neural ideals, then how are C and $\mathcal{D}$ related?

Recall in Definition 2.45 that we used a surjective function $\eta : [n] \to [m]$ to define a map between the polynomial rings $R[n] \to R[m]$ by applying η to the indices on each variable. We can likewise use η to define a transformation of codes on n bits to codes on m bits.

Definition 2.52. Let $\eta : [n] \to [m]$ be a surjective function. The *code identification map* induced by η is a function η' mapping codes $C \subseteq \mathbb{F}_2^n$ to codes $\mathcal{D} \subseteq \mathbb{F}_2^m$ defined so that

$$\eta'(C) := \{v \in \mathbb{F}_2^m \mid \eta^{-1}(\text{supp}(v)) = \text{supp}(c) \text{ for some } c \in C\}.$$

Notice that when $n = m$ this has the effect of simply permuting indices in C according to η. It is important to note that this map sends codes to codes, but is not a well defined function on the individual codewords. At first this definition may seem strange, so it is worth considering a small example.

Example 2.53. Let $\eta : [4] \to [3]$ be the function so that $4 \mapsto 3$, $3 \mapsto 3$, $2 \mapsto 2$ and $1 \mapsto 1$. Let C be the code on four bits defined as

$$C = \left\{ \begin{matrix} 0000,\ 1000,\ 0010,\ 1010, \\ 1100,\ 1011,\ 1110,\ 1111 \end{matrix} \right\}.$$

Then

$$\eta'(C) = \left\{ \begin{matrix} 000,\ 100,\ 101, \\ 110,\ 111 \end{matrix} \right\}.$$

Above we have highlighted in red the codewords in C which are preimages under η of codewords in $\mathcal{D}$. Note that the last two bits in all of the highlighted codewords must agree since $\eta(3) = \eta(4) = 3$.

As one would hope, identification maps between the polynomial rings and identification maps between codes are related naturally. To establish this relationship we must first prove the following lemma.

Lemma 2.54. *Let $\rho_v \in R[m]$ be an indicator polynomial, let $\eta : [n] \to [m]$ be a surjective function, and let $u \in \mathbb{F}_2^n$ be such that $\mathrm{supp}(u) = \eta^{-1}(\mathrm{supp}(v))$. Then ρ_u is the unique indicator polynomial for which $\rho_v = \eta(\rho_u)$.*

Proof. Let u be any vector in $\mathbb{F}_2^n$. Then we have directly that

$$\begin{aligned} \eta\left(\rho_u\right) &= \eta\left(\prod_{i \in \mathrm{supp}(u)} x_i \prod_{j \in [n] \setminus \mathrm{supp}(u)} (1 - x_j) \right) \\ &= \prod_{i \in \mathrm{supp}(u)} x_{\eta(i)} \prod_{j \in [n] \setminus \mathrm{supp}(u)} (1 - x_{\eta(j)}) \\ &= \prod_{i \in \eta(\mathrm{supp}(u))} x_i \prod_{j \in \eta([n] \setminus \mathrm{supp}(u))} (1 - x_j). \end{aligned}$$

If $\mathrm{supp}(u) = \eta^{-1}(\mathrm{supp}(v))$ then we have $\eta(\mathrm{supp}(u)) = \mathrm{supp}(v)$ and $\eta([n] \setminus \mathrm{supp}(u)) = [m] \setminus \mathrm{supp}(v)$, and so $\eta(\rho_u) = \rho_v$.

To see that this is the unique case where $\eta(\rho_u) = \rho_v$, suppose that $\mathrm{supp}(u) \neq \eta^{-1}(\mathrm{supp}(v))$. Then either $\eta(\mathrm{supp}(u)) \neq \mathrm{supp}(v)$, in which case

$\eta(\rho_u)$ is not divisible by the correct variables, or $\eta(\text{supp}(u)) = \text{supp}(v)$, but $\eta([n] \setminus \text{supp}(u))$ contains some $k \in \text{supp}(v)$. Then $\eta(\rho_u)$ is divisible by $x_k(1 - x_k) = 0$, and must be zero itself. We conclude that $\eta(\rho_u) = \rho_v$ if and only if $\text{supp}(u) = \eta^{-1}(\text{supp}(v))$. □

With this lemma established we can completely classify the correspondence between identification maps of polynomial rings and identification maps between collections of codes.

Lemma 2.55. *Let $\eta : R[n] \to R[m]$ be an identification map induced by the surjective function $\eta : [n] \to [m]$, and let η' be the code identification map induced by η. Then*

$$\eta(J_C) = J_{\eta'(C)}$$

for any code $C \subseteq \mathbb{F}_2^n$.

Proof. We have observed previously that η is a map respecting neural ideals, so $\eta(J_C)$ is the neural ideal of some code. We aim to show that this code is precisely $\eta'(C)$. To prove this it suffices to show that an indicator polynomial ρ_v is in $\eta(J_C)$ if and only if it vanishes on $\eta'(C)$, or equivalently if and only if $v \notin \eta'(C)$.

First suppose that ρ_v is in $\eta(J_C)$, and let u be such that $\text{supp}(u) = \eta^{-1}(\text{supp}(v))$. We know by Lemma 2.54 that ρ_u must be in J_C, and in particular $u \notin C$. But then $v \notin \eta'(C)$ by definition of η'. For the converse, suppose $v \notin \eta'(C)$. Then clearly $u \notin C$, and so $\rho_u \in J_C$. Since $\eta(\rho_u) = \rho_v$ we have $\rho_v \in \eta(J_C)$ as desired. This proves the result. □

We next turn to bit flipping maps, and obtain a similar result. In the definition below, the vector $e^{(i)}$ is the i-th standard basis vector for $\mathbb{F}_2^n$, which has a 1 in index i and zeroes elsewhere.

Definition 2.56. Let i be any index in $[n]$. The i-th *bit flipping code transformation* is a function δ_i mapping codes on n bits to codes on n bits defined by

$$\delta_i(C) = \{c + e^{(i)} \mid c \in C\}.$$

Recall that the i-th bit flipping map δ_i is defined between polynomial rings by sending $x_i \mapsto 1 - x_i$ (cf. Definition 2.30). As one would hope, this algebraic map and the transformation between codes are compatible. The lemma below makes this correspondence precise.

Lemma 2.57. *Let $C \subseteq \mathbb{F}_2^n$ be a code, and let $\mathcal{D} = \delta_i(C)$. Then $\delta_i(J_C) = J_{\mathcal{D}}$.*

Proof. We know that δ_i respects neural ideals, and so $\delta_i(J_C)$ will be the neural ideal of some code. To show that this code is $\mathcal{D}$ it suffices to show that $\rho_v \in \delta_i(J_C)$ if and only if $v \notin \mathcal{D}$. Let ρ_v be an indicator in $\delta_i(J_C)$. Notice that $\delta_i(\rho_{v+e^{(i)}}) = \rho_v$ and since δ_i is an automorphism we must have $\rho_{v+e^{(i)}} \in J_C$. In particular, $\rho_v \in \delta_i(J_C)$ if and only if $\rho_{v+e^{(i)}} \in J_C$. Equivalently, $\rho_v \in \delta_i(J_C)$ if and only if $v + e^{(i)} \notin C$. But $v + e^{(i)} \notin C$ precisely when $v + 2e^{(i)} = v \notin \mathcal{D}$, since if v were in $\mathcal{D}$ then $v + e^{(i)}$ would be in C. We conclude that $\rho_v \in \delta_i(C)$ if and only if $v \notin \mathcal{D}$ as desired. □

Notice that adding $e^{(i)}$ to the vectors in C has the effect of "flipping" the i-th bit in every vector: since we are working over $\mathbb{F}_2$ any vector with a 1 in index i will become the same vector but with a 0 in index i when we add $e^{(i)}$. Likewise, any vector with a 0 in index i becomes the same vector with a 1 in that index.

Finally, we turn our attention to restriction maps $\omega : R[n] \to R[m]$. To characterize the effect of these maps on codes we introduce the notion of "restricting" a code to codewords satisfying certain conditions.

Definition 2.58. Let $C \subseteq \mathbb{F}_2^n$ be a code, let m and m' be integers so that $1 \leq m \leq m' \leq n$, and let $\sigma = [n] \setminus [m']$. Then the *restriction* of C to (m, m') is the code

$$\text{rest}(C, m, m') := \{\tau \subseteq [m] \mid \tau \cup \sigma \in C\} \subseteq \mathbb{F}_2^m.$$

In the definition above we describe the codewords in $\text{rest}(C, m, m')$ in terms of their supports. Intuitively, the restriction is the result of collecting all codewords in C which have 0 in the indices between m and m', and 1 in the indices between m' and n, and then "forgetting" all indices except those between 1 and m. This process warrants a concrete example.

Example 2.59. Consider the code on 8 bits given below. We give its restriction with $m = 3$ and $m' = 6$. The resulting code is on 5 bits, and comes from the "compatible" codewords highlighted in red. These codewords have 0 in indices 4,5, and 6, and 1 in the indices 7 and 8, as dictated by our choice of m and m'. To obtain the restriction we forget the last five indices in the highlighted codewords, since these indices are fixed and do not contain any

interesting information.

$$C = \begin{Bmatrix} 00000000 \\ 00100011 \\ 01011100 \\ 11111101 \\ 10100011 \\ 10010011 \\ 10011011 \\ 11100011 \\ 01100011 \end{Bmatrix} \qquad \text{rest}(C, 3, 6) = \begin{Bmatrix} 001 \\ 101 \\ 111 \\ 011 \end{Bmatrix}$$

In the lemmas below we complete characterize the induced map of any restriction homomorphism in terms of restrictions of codes.

Lemma 2.60. *Let f be a pseudomonomial and let g be any polynomial in $R[n]$. Then $gf = g'f$ where g' is a polynomial that does not depend on any variables present in f.*

Proof. Write $f = x_\sigma \overline{x}_\tau$ and write g as a sum of monomials:

$$g = \sum x_\gamma.$$

Then notice that for any γ we either have $x_\gamma f = 0$ (when $\gamma \cap \tau \neq \emptyset$) or $x_\gamma f = x_{\gamma'} f$ where γ' is disjoint from σ and τ, since $x_i^2 = x_i$ in $R[n]$. Distributing f into each term of g, we have

$$gf = \sum x_\gamma x_\sigma \overline{x}_\tau = \sum x_{\gamma'} x_\sigma \overline{x}_\tau = \left(\sum x_{\gamma'}\right) f = g'f.$$

Clearly g' cannot depend on variables present in f since each γ' is disjoint from σ and τ.

□

Lemma 2.61. *Let $1 \leq m \leq m' \leq n$ and let $\omega_{m,m'} : R[n] \to R[m]$ be a restriction map as in Definition 2.28. Then*

$$\omega_{m,m'}(J_C) = J_{\text{rest}(C,m,m')}.$$

Proof. For the sake of avoiding clutter we will drop the subscript on the homomorphism and write ω for $\omega_{m,m'}$. We know that $\omega(J_C)$ is a neural ideal since ω respects neural ideals. It thus suffices to show that the canopies of $\omega(J_C)$ and $J_{\text{rest}(C,m,m')}$ are equal. First, let $\sigma = \{m'+1, m'+2, \ldots, n\}$

and let $\tau = \{m+1, m+2, \ldots, m'\}$. The indices $i \in \sigma$ are those for which $\omega(x_i) = 1$ and the indices $j \in \tau$ are those for which $\omega(x_j) = 0$. Then define a pseudomonomial $f = x_\sigma \overline{x}_\tau$ and notice that $\omega(f) = 1$. Let ρ_v be any indicator in the canopy of $\omega(J_C)$, and let $\hat{f} \in J_C$ be any polynomial such that $\omega(\hat{f}) = \rho_v$. Now notice that $f\hat{f} \in J_C$ and furthermore that

$$\omega(f\hat{f}) = \omega(f)\omega(\hat{f}) = \omega(f) = \rho_v.$$

By Lemma 2.60 we can write $f\hat{f} = f'\hat{f}$ where f' does not depend on any variable in σ or τ. Then $\omega(f') = \rho_v$. But all variables that f' depends on are fixed by ω, so we actually have $f' = \omega(f') = \rho_v$. As a result we conclude that $\rho_v \in J_C$. Letting $\alpha = \text{supp}(v)$ and $\beta = [m] \setminus \text{supp}(v)$ we can write $\rho_v = x_\alpha \overline{x}_\beta$, and we see that the indicator polynomial

$$f\rho_v = x_{\sigma\cup\alpha}\overline{x}_{\tau\cup\beta}$$

is in J_C. This is the indicator for the vector which results from "padding" v by $m' - m$ zeros and then $n - m'$ ones. We then conclude that an indicator ρ_v is in $\omega(J_C)$ if and only if $f\rho_v$ is in J_C.

By the above results it suffices to show that for any $v \in \mathbb{F}_2^m$, the indicator $f\rho_v$ is in J_C if and only if $v \notin J_{\text{rest}(C,m,m')}$. The indicator $f\rho_v$ is in J_C if and only if the vector with support $\text{supp}(v) \cup \sigma$ is not in C. By the definition of the restriction this occurs if and only if $v \notin \text{rest}(C, m, m')$. This proves the desired result. □

We thus see that applying restriction maps corresponds to restricting our code to a certain set of compatible codewords. We have now described the effects of all possible homomorphisms respecting neural ideals on codes. The following diagram summarizes the correspondence between maps respecting neural ideals and the transformations that these maps induce on codes. This correspondence is bijective, since any map respecting neural ideals induces a unique transformation of codes. Although permutations are redundant to identification maps we include them in our summary since they are natural and relatively simple. In the diagram below algebraic transformations are on the left and the effects on codes are given on the right.

Permutations $x_i \mapsto x_{\lambda(i)}$ for a permutation λ	$\longleftrightarrow$	Permutation of indices in code $C \mapsto \lambda(C)$
Identification maps $x_i \mapsto x_{\eta(i)}$ for a surjective function $\eta : [n] \to [m]$	$\longleftrightarrow$	Identification given by $C \mapsto \eta'(C)$
Bit flipping maps $x_i \mapsto 1 - x_i$	$\longleftrightarrow$	Adding $e^{(i)}$ to all vectors in code
Restriction maps $\omega_{m,m'}$	$\longleftrightarrow$	Restriction given by $C \mapsto \text{rest}(C, m, m')$

By Theorem 2.3 every map respecting neural ideals can be expressed as a composition of the maps on the left, and so we have described the effect of any map respecting neural ideals on the codes corresponding to those ideals. In particular, we have answered Question 2.51.

These maps have several immediately natural uses. Permutation can be used to rearrange the bits in a code to a more convenient or easily readable manner without changing the fundamental structure of the code, and bit flipping corresponds to negating the behavior of a particular neuron. More surprisingly, restriction can be used to compute neural ideals for links of a code. Recall from Definition 1.14 that for any code we can compute the link of a set $\sigma \subseteq [n]$ in C. The link can encode important information about the code it is derived from, and serves as a tool in classifying the convexity of some codes, as in Curto et al. (2015). We will examine these properties

in depth later, but for now we show that the link is in fact a special case of permutation maps composed with restriction maps. This allows us to determine the structure of the neural ideal of the link simply by applying a homomorphism to a generating set for J_C.

Lemma 2.62. *Let $C \subseteq \mathbb{F}_2^n$ be a code, let $m \in [n]$, and let $\sigma = \{i \in [n] \mid m < i \leq n\}$. Then $\mathrm{Lk}_\sigma(C) = \mathrm{rest}(C, m, m)$.*

Proof. Let $\tau \in \mathbb{F}_2^m$ be a vector as defined by its support. Then $\tau \in \mathrm{Lk}_\sigma(C)$ if and only if $\tau \cup \sigma$ is in C. But $\sigma = \{m+1, m+2, \ldots, n\}$ and so τ is in $\mathrm{rest}(C, m, m)$ under precisely the same conditions. □

This provides a convenient computational tool for determining the canonical form of the link of a code. To compute the canonical form of $\mathrm{Lk}_\sigma(C)$ for any $\sigma \subseteq [n]$ we need only compose a restriction with a permutation which sends σ to the last indices in $[n]$, and then undo the permutation.

More generally, the results of this section demonstrate that maps preserving neural ideals can help us understand relationships between canonical forms of related codes. In the next section we consider a slightly more general class of maps: $\mathbb{F}_2$ linear transformations of $R[n]$ which preserve neural ideals. Although these maps do not preserve the ring structure of $R[n]$ in general, we will see that they still have some interesting properties related to neural ideals.

2.4.5 Linear transformations preserving neural ideals

So far we have considered only the ring structure of $R[n]$. However, one can also treat this as an $\mathbb{F}_2$-vector space. One natural basis for this space is the collection of squarefree monomials in $R[n]$. However, Lemma 2.14 tells us that each of these monomials can be obtained as the sum of all indicator polynomials it divides. As a result, another basis for $R[n]$ over $\mathbb{F}_2$ is the collection of all indicator polynomials. We have seen that each neural ideal can be uniquely associated to a collection of indicator polynomials, and so this basis serves as a natural starting point for analyzing neural ideals from a linear algebraic perspective.

It is not obvious that a linear algebraic approach will yield any information about neural ideals, since neural ideals are associated to the ring structure of $R[n]$. However, the properties of pseudomonomials discussed in Section 2.2 allow us to draw some useful correspondences. In particular, the following lemma shows that, for indicator polynomials, spanning a subspace is the same as generating it as an ideal.

Lemma 2.63. *Let $v_1, v_2, \ldots, v_k$ be vectors in $\mathbb{F}_2^n$. Then*

$$\langle \rho_{v_1}, \rho_{v_2}, \ldots, \rho_{v_k} \rangle_{R[n]} = \operatorname{span}_{\mathbb{F}_2} \left(\rho_{v_1}, \rho_{v_2}, \ldots, \rho_{v_k} \right).$$

Proof. The inclusion $\langle \rho_{v_1}, \rho_{v_2}, \ldots, \rho_{v_k} \rangle_{R[n]} \supseteq \operatorname{span}_{\mathbb{F}_2} \left(\rho_{v_1}, \rho_{v_2}, \ldots, \rho_{v_k} \right)$ is immediate since $\mathbb{F}_2 \subset R[n]$. We must then show that any polynomial f in the ideal $\langle \rho_{v_1}, \rho_{v_2}, \ldots, \rho_{v_k} \rangle$ can be obtained as an $\mathbb{F}_2$-linear combination (i.e., sum) of the various ρ_{v_i}. It suffices to show that for any indicator ρ_{v_i}, all multiples of ρ_{v_i} are in $\operatorname{span}_{\mathbb{F}_2} \left(\rho_{v_1}, \rho_{v_2}, \ldots, \rho_{v_k} \right)$.

To prove this, first let $\sigma = \operatorname{supp}(v_i)$ and $\tau = [n] \setminus \operatorname{supp}(v_i)$. We know that $\rho_{v_i} = x_\sigma \overline{x}_\tau$. Then let x_γ for $\gamma \subseteq [n]$ be any squarefree monomial. If $\gamma \cap \tau$ is nonempty, then $x_\gamma \rho_{v_i}$ has a factor of $x_j \overline{x}_j$ for some j, and is zero. Otherwise, $\gamma \subseteq \sigma$ and we have that

$$\begin{aligned} x_\gamma \rho_{v_i} &= x_\gamma x_\sigma \overline{x}_\tau \\ &= (x_\gamma)^2 x_{\sigma \setminus \gamma} \overline{x}_\tau \\ &= x_\gamma x_{\sigma \setminus \gamma} \overline{x}_\tau && \text{Since } x_j^2 = x_j \text{ in } R[n]. \\ &= x_\sigma \overline{x}_\tau \\ &= \rho_{v_i}. \end{aligned}$$

Now, any polynomial in $R[n]$ can be written as the sum of squarefree monomials. By distributivity, the product of any polynomial with ρ_{v_i} will just be the sum of ρ_{v_i} with itself and zero some number of times. This is clearly in $\operatorname{span}_{\mathbb{F}_2} \left(\rho_{v_1}, \rho_{v_2}, \ldots, \rho_{v_k} \right)$. We conclude that any polynomial in the ideal generated by the various ρ_{v_i} is spanned by the ρ_{v_i} over $\mathbb{F}_2$, proving the desired result. □

Corollary 2.64. *Neural ideals in $R[n]$ are precisely the $\mathbb{F}_2$-subspaces of $R[n]$ with a basis consisting of indicator polynomials.*

This somewhat surprising fact allows us to study neural ideals from a purely linear algebraic perspective. On the one hand, this perspective is extremely simplistic and forgets a great deal of information, since vector spaces over $\mathbb{F}_2$ are not rich in structure. On the other hand, this simplicity can be of use in computing and understanding neural ideals. We will say that a linear transformation respects neural ideals if $T(J_C)$ is a neural ideal for any code C.

Lemma 2.65. *A linear map $T : R[n] \to R[m]$ respects neural ideals if and only if it maps indicator polynomials to indicator polynomials or zero.*

Proof. ($\Rightarrow$) For any vector $v \in \mathbb{F}_2^n$ consider the neural ideal $\langle \rho_v \rangle$. By Corollary 2.64 this ideal is precisely the set $\{\rho_v, 0\}$. The image of this ideal under T is then either $\{0\}$ or a neural ideal consisting of two elements. If the image under T is zero then clearly $T(\rho_v) = 0$. Otherwise $T(\rho_v)$ must be an indicator, since otherwise $\{T(\rho_v), 0\}$ would not be a neural ideal in $R[m]$. Thus T maps each indicator to either zero or some other indicator.

($\Leftarrow$) By Corollary 2.64 we have that any neural ideal J_C is spanned over $\mathbb{F}_2$ by a set of indicators $\{\rho_{v_1}, \rho_{v_2}, \ldots, \rho_{v_k}\}$. The image of such a neural ideal under T will be spanned by $\{T(\rho_{v_1}), T(\rho_{v_2}), \ldots, T(\rho_{v_k})\}$. Each $T(\rho_{v_i})$ is either zero or an indicator, and so $T(J_C)$ is spanned by a set of indicators. Again by Corollary 2.64 this implies that $T(J_C)$ is a neural ideal, proving the desired result. □

A natural hope is that linear transformations preserve neural ideals in similar ways to ring homomorphisms, or perhaps that they can be useful in determining the canonical forms of related codes. The following definition makes this notion more precise.

Definition 2.66. Let $T : R[n] \to R[m]$ be a linear transformation that respects neural ideals. Then T *preserves canonical forms* if for every code C we have

$$T(\mathrm{CF}(J_C)) = \mathrm{CF}(T(J_C)).$$

That is, if T commutes with taking the canonical form.

Unfortunately, linear transformations are too general of a structure to always preserve the canonical form in a convenient way. The next lemma shows that linear transformations that preserve canonical form must come from graph homomorphisms of hypercubes. For the following lemma, recall that we can treat $\mathbb{F}_2^n$ as an n-dimensional hypercube graph by letting two vectors u and v be adjacent if they differ in one index.

Theorem 2.4. *Let $T : R[n] \to R[m]$ be a linear transformation that respects neural ideals and which does not map any indicator to zero. If T preserves canonical forms then it induces a graph homomorphism $\phi : \mathbb{F}_2^n \to \mathbb{F}_2^m$ so that $T(\rho_v) = \rho_{\phi(v)}$.*

Proof. Suppose that T respects neural ideals and preserves canonical forms, and does not map any indicator to zero. Then T maps indicators to indicators by Lemma 2.65, and so T induces a map $\phi : \mathbb{F}_2^n \to \mathbb{F}_2^m$ given by $\phi(v) = u$ where $T(\rho_v) = \rho_u$.

We aim to show that ϕ is a graph homomorphism. Let v and u be vectors that differ in exactly one bit, and hence are adjacent in $\mathbb{F}_2^n$ considered as a

graph. We know that $\rho_v + \rho_u = f$ is a pseudmonomial, consisting of factors for all the bits which are the same in v and u. Furthermore, the neural ideal $\langle \rho_v, \rho_u \rangle$ has canonical form exactly $\{f\}$. Now consider T applied to this ideal. We obtain a new neural ideal $\langle T(\rho_v), T(\rho_u) \rangle$. By assumption, $\{T(f)\}$ is the canonical form of this ideal. But elements of the canonical form are always obtained as sums of the indicators in the ideal, in this case $T(\rho_v)$ and $T(\rho_u)$. Since T does not send either ρ_u or ρ_v to zero it must be the case that the sum of these two indicators is a pseudomonomial. This happens only if $T(\rho_v)$ and $T(\rho_u)$ are indicators for vectors that differ in a single bit. That is, if $\phi(v)$ and $\phi(u)$ are adjacent in $\mathbb{F}_2^m$. Thus ϕ is a graph homomorphism. □

Linear maps induced by graph homomorphisms are not an immediately natural notion in the context of neural codes, and this lemma suggests that perhaps we have strayed too far in choosing to forget the ring structure of $R[n]$. Nevertheless, linear transformations are relatively simple to work with computationally, and so perhaps future work in this direction could be useful.

Notice that throughout this section we have not spoken of the effects of these linear transformations on the codes themselves. When we considered homomorphisms that respect neural ideals, we were able to draw a nice correspondence between homomorphisms and transformations of codes. In the case of linear transformations that respect neural ideals, the code transformations induced do not have natural properties or descriptions. In part, this arises from the fact that we consider the action of the linear transformation on indicators, and these indicators correspond to the complement of a code, not the code itself. Furthermore, these transformations can send indicators to any other indicator. This level of freedom in the linear transformations makes it difficult to guarantee any nice structure regarding the codes themselves.

2.5 Algebraic Transformations and the Canonical Form

In the previous sections we have examined homomorphisms between polynomial rings $\mathbb{F}_2[n]$, and their relationship to codes via neural ideals. In Section 2.4.2 we completely classified all homomorphisms which respect neural ideals and in Section 2.4.3 we expanded this classification to homomorphisms modulo the Boolean relations $x_i\overline{x}_i$. Finally, in Section 2.4.4 we

characterized the effects of these algebraic maps on codes via the correspondence between neural ideals and codes. At the end of this discussion we considered linear maps, and their relationship to the canonical form (cf. Definition 2.21). This section will focus on understanding how the canonical form changes under the homomorphisms described in Section 2.4.2 and Section 2.4.3.

For any homomorphism ϕ respecting neural ideals, we aim to compute $\mathrm{CF}(\phi(J_\mathcal{C}))$ from $\mathrm{CF}(J_\mathcal{C})$ and knowledge of ϕ. We will begin by considering the homomorphisms $\phi : \mathbb{F}_2[n] \to \mathbb{F}_2[m]$, which we discussed in Section 2.4.2. We will see later that these behave much better with respect to the canonical form than homomorphisms modulo Boolean relations. A first hope is that we can simply apply our homomorphism ϕ to each polynomial in $\mathrm{CF}(J_\mathcal{C})$ to obtain $\mathrm{CF}(\phi(J_\mathcal{C}))$. That is, we would hope that $\phi(\mathrm{CF}(J_\mathcal{C})) = \mathrm{CF}(\phi(J_\mathcal{C}))$. However this is unfortunately not the case, as demonstrated by the following example.

Example 2.67. Consider the neural ideal $J_\mathcal{C} = \langle x_1x_2, x_1x_3\rangle$ and the restriction homomorphism $\phi : \mathbb{F}_2[3] \to \mathbb{F}_2[2]$ which sends $x_3 \mapsto 1$. The canonical form of J is just $\{x_1x_2, x_1x_3\}$ and its image under ϕ is $\{x_1x_2, x_1\}$. This is clearly not the canonical form of $\phi(J_\mathcal{C})$ since x_1 divides x_1x_2, and hence x_1x_2 is not minimal in $\phi(J_\mathcal{C})$.

In the above example, applying the homomorphism ϕ to $\mathrm{CF}(J_\mathcal{C})$ introduces redundancy in the form of x_1x_2 and x_1 both being present. We will see that this effectively the only thing that can go wrong when we consider homomorphisms without modding out by Boolean relations. First, we formalize the notion of redundancy.

Definition 2.68. Let $C \subseteq \mathbb{F}_2[n]$ be a set of pseudomonomials. We say that C is *redundancy free* if it does not contain any pseudomonomials which divide one another. Otherwise, we say that C is *redundant*.

Note that the canonical form is always redundancy free. On the other hand, a redundancy free set of pseudomonomials need not be a canonical form (recall Example 2.22). As we saw in Example 2.67, the image of a canonical form under a homomorphism respecting neural ideals can be redundant. The following lemma show that things can get no worse than this.

Lemma 2.69. *Let $\phi : \mathbb{F}_2[n] \to \mathbb{F}_2[m]$ be a homomorphism respecting neural ideals and let $J_\mathcal{C}$ be a neural ideal. Then $\mathrm{CF}(\phi(J_\mathcal{C})) \subseteq \phi(\mathrm{CF}(J_\mathcal{C}))$. Equivalently, if $f \in \mathrm{CF}(\phi(J_\mathcal{C}))$ then there exists a pseudomonomial $\hat{f} \in \mathrm{CF}(J_\mathcal{C})$ so that $\phi(\hat{f}) = f$.*

Proof. By Theorem 2.2, every homomorphism respecting neural ideals can be written as the composition of a bit flipping map, permutation, and restriction. We will show that the lemma is true for these three classes, and it will follow that it holds for any general homomorphism respecting neural ideals. Permutation is obvious since it only changes the labels on variables, and cannot affect minimality of pseudmonomials with respect to division. Indeed, we have more strongly that for any permutation λ and code C,

$$\lambda(\mathrm{CF}(J_C)) = \mathrm{CF}(\lambda(J_C)).$$

The case of bit flipping maps is similar. Suppose that $f \in \mathrm{CF}(\delta_i(J_C))$. Then since δ_i is an involution we know that $\delta_i(f) \in J_C$. Furthermore, if $g \neq \delta_i(f)$ divides $\delta_i(f)$ for a pseudomonomial $g \in J_C$, then $\delta_i(g) \neq f$ divides f, and hence would be in $\mathrm{CF}(\delta_i(J_C))$. We conclude that $\hat{f} = \delta_i(f)$ is in $\mathrm{CF}(J_C)$, and since $\delta_i(\hat{f}) = f$ the desired result holds.

This leaves the case that our homomorphism is a restriction $\omega : \mathbb{F}_2[n] \to \mathbb{F}_2[m]$. Let $f \in \mathrm{CF}(\phi(J_C))$. We aim to find a pseudomonomial in $\mathrm{CF}(J_C)$ whose image is f. First we prove that J_C contains a pseudomonomial that maps to f. We know there exists some polynomial in J_C mapping to f, say g. Since $g \in J_C$ we can write it as

$$g = \sum g_v \rho_v$$

where the sum above is over some collection of vectors *not* in C, and each g_v is simply a polynomial in $\mathbb{F}_2[n]$. From this sum, we will remove any sub-sum which is in the kernel of ϕ. This yields a polynomial g' whose image under ϕ is clearly still f, and which can be expressed as

$$g' = \sum g'_v \rho_v.$$

Since none of the terms in the above sum combine to a polynomial in the kernel of ϕ, we must have $\phi(g'_v) = 1$ for all g'_v: otherwise, the image of g' under ϕ would have degree larger than n. Hence, we obtain a polynomial

$$g'' = \sum \rho_v$$

where $\phi(g'') = f$. Note that $g'' \in J_C$ since each ρ_v is in J_C. We now claim that g'' is a pseudomonomial. Since ϕ respects neural ideals and $\phi(g'') = f$ we have that

$$\sum \phi(\rho_v) = \phi(g'') = f = \sum_{f | \rho_u} \rho_u$$

where above we have leveraged the fact that f is the sum of all pseudomonomials in its canopy. Since pseudomonomials form a linearly independent set over $\mathbb{F}_2$ these sums must be equal term by term, and hence each ρ_v is a preimage of a pseudomonomial in the canopy of f.

We next use the fact that f is a projection to completely describe the ρ_v which sum to g. Since ϕ is a restriction, we can define the following sets of indices:

$$a_0 = \{i \in [n] \mid \phi(x_i) = 0\}, \text{ and}$$
$$a_1 = \{i \in [n] \mid \phi(x_i) = 1\}.$$

We know that if i is an index not in either of these sets, then $\phi(x_i) = x_i$. We can also notice that, of the pseudmonomials whose variables are indexed by a_0 and a_1 there is exactly one which is not in the kernel of ϕ, namely

$$x_{a_0}\overline{x}_{a_1}.$$

Call this pseudomonomial h, and notice that every ρ_v must have it as a factor, since otherwise some factor in ρ_v would vanish. Also notice that $\phi(h) = 1$, and $\phi(\rho_v) = \rho_u$ where ρ_u is one of the terms in the sum making up f. That is, we can write $\rho_v = h\rho_u$ for all ρ_v. Hence we have

$$g'' = \sum \rho_v = \sum h\rho_u = h \sum \rho_u = hf.$$

Since h and f are pseudomonomials depending on different indices, their product is clearly a pseudomonomial. We conclude that $g'' \in J_C$ is a pseudomonomial which maps to f under ϕ.

Now, consider the set of all pseudomonomials in J_C which map to f under ϕ. Let $\hat{f}$ be such a pseudomonomial which is minimal with respect to division. We know such a $\hat{f}$ exists since we have just shown that there is at least one pseudomonomial in J_C mapping to f. We now claim that $\hat{f} \in \mathrm{CF}(J_C)$. Suppose towards a contradiction that this were not true. Then there would exist some $\hat{g}$ properly dividing $\hat{f}$ in J_C. Since $\hat{g}$ divides $\hat{f}$ it is clear that $\phi(\hat{g})$ divides $\phi(\hat{f}) = f$. But f is minimal in $\phi(J_C)$ and hence $\phi(\hat{g}) = f$. This implies that $\hat{g}$ is in the collection of pseudomonomials in J_C mapping to f, contradicting the minimality of $\hat{f}$ in this set. Therefore $\hat{f}$ is in $\mathrm{CF}(J_C)$ are desired. □

This lemma tells us that, to compute the canonical form of $\phi(J_C)$, we need only apply ϕ to every polynomial in $\mathrm{CF}(J_C)$ and remove redundant elements

with respect to division. Note that we have relied on the fact that we are *not* working mod the Boolean relations by specifically considering the different cases that can occur. We will see in a later example that working mod the Boolean relations changes things significantly. In Algorithm 1 we provide a straightforward method of removing redundancy from a collection of pseudomonomials.

Algorithm 1 Algorithm to remove redundancy from a set of pseudomonomials C

```
1: procedure RemoveRedundancy(C)
2:     Let C′ be an empty set
3:     for each pseudomonomial f in C do
4:         for each pseudomonomial f′ in C′ do
5:             If f′ is a multiple of f, delete it from C′
6:             If f is a multiple of f′, set f := f′
7:         Add f to C′ if it is not already present.
8:     Return C′
```

Note immediately that Algorithm 1 returns a set which generates the same ideal as C, since it simply returns the collection of minimal elements in C with respect to division.

We next turn our attention to the more general case of homomorphisms $\phi : R[n] \to R[m]$ which preserve neural ideals. Our work above completely described how bit flipping, permutation, and restriction affect canonical forms. The same arguments given before hold modulo the Boolean relations, and so our main task is to deal with the fourth possiblity described in Theorem 2.3: identification maps. To begin with, we will show that identification does not behave as nicely as the other maps, in that it can introduce more than just redundancy.

Example 2.70. Let J_C be the ideal whose canonical form is $\langle x_1x_2, x_3\overline{x}_4\rangle$, and let $\phi : R[4] \to R[2]$ be the identification map which sends $x_3 \mapsto x_1$ and $x_4 \mapsto x_2$ while keeping x_1 and x_2 fixed. Then clearly

$$\phi(\mathrm{CF}(J_C)) = \{\phi(x_1x_2), \phi(x_3\overline{x}_4)\} = \{x_1x_2, x_1\overline{x}_2\}.$$

Notice that this set is redundancy free, but cannot be a canonical form since the sum of x_1x_2 and $x_1\overline{x}_2$ is x_1, and hence neither of these would be minimial in an ideal generated by the two.

The example above tells us that we *cannot* simply consider the image of a canonical form under an arbitrary homomorphism $\phi : R[n] \to R[m]$ respecting neural ideals and remove redundancy to obtain a new canonical form. However, observe that if

$$J_C = \langle f_1, \ldots, f_k \rangle$$

in $R[n]$ then

$$\phi(J_C) = \langle \phi(f_1), \ldots, \phi(f_k) \rangle.$$

Indeed, any surjective homomorphism will preserve generating sets for ideals. As such, we will describe a general algorithm for computing the canonical form from a generating set of pseudmonomials. This will allow for the following approach to computing $\mathrm{CF}(\phi(J_C))$ for an arbitrary homomorphism respecting neural ideals $\phi : R[n] \to R[m]$:

$$\text{Start with } \mathrm{CF}(J_C) \quad \longrightarrow \quad \begin{array}{c}\text{Compute the}\\ \text{generating set}\\ \phi(\mathrm{CF}(J_C)) \text{ for } \phi(J_C)\end{array} \quad \longrightarrow \quad \begin{array}{c}\text{Use this generating set}\\ \text{to compute } \mathrm{CF}(\phi(J_C))\end{array}$$

Before describing the algorithm for the final step above we develop some further notation related to pseudomonomials. Throughout this section we will consider pseudomonomials as partially ordered with respect to division. When we speak of a "minimal" pseudomonomial in a set we will always mean that no divisor of it is in the set.

Definition 2.71. Let $f, g \in \mathbb{F}_2[n]$ be pseudomonomials. We say that f and g are *compatible* if there exists a unique i so that x_i divides f and $\overline{x}_i$ divides g, or so that $\overline{x}_i$ divides f and x_i divides g.

Compatibility gives us a way to combine pseudomonomials to create new pseudomonomials, and this combination process will be used heavily in our algorithm.

Definition 2.72. Let $f, g \in \mathbb{F}_2[n]$ be compatible pseudomonomials so that x_i divides f and $\overline{x}_i$ divides g. We define the *composite* of f and g to be the product of all factors other than x_i and $\overline{x}_i$ which appear in at least one of f and g. The composite is denoted $f \oplus g$.

Example 2.73. Consider the pseudomonomials $x_1x_2x_3$ and $x_1\overline{x}_3x_4\overline{x}_5$, which are compatible since they disagree only in their x_3 term. The composite of these pseudomonomials will be $x_1x_2x_4\overline{x}_5$.

Lemma 2.74. *Let $f, g \in \mathbb{F}_2[n]$ be compatible pseudomonomials. Then the pseudomonomial $f \oplus g$ is in $\langle f, g \rangle$.*

Proof. Without loss of generality suppose that x_i divides f and $\overline{x}_i$ divides g, but the same is not true of any other variable. Let $h = f \oplus g$ and notice that $x_i h$ is a multiple of f and $\overline{x}_i h$ is a multiple of g. That is, there exist f' and g' so that $ff' = x_i h$ and $gg' = \overline{x}_i h$. Then we have that

$$h = (x_i + \overline{x}_i)h = x_i h + \overline{x}_i h = ff' + gg' \in \langle f, g \rangle$$

proving the desired result. □

Next we present an algorithm for computing the canonical form of a neural ideal from a generating set of pseudomonomials. Recall from Theorem 2.1 that neural ideals are precisely those which have a generating set consisting of pseudomonomials. Given the utility of the canonical form, we seek to easily compute the canonical form from any generating set of pseudomonomials (for example, a set of indicator pseudomonomials as described in Definition 2.7, or the image of a canonical form under some homomorphism). In Algorithm 2 we describe exactly such a procedure.

Algorithm 2 Algorithm to convert a generating set to the canonical form

```
1:  procedure ComputeCF(C)
2:      Set C := RemoveRedundancy(C)
3:      Let C′ be an empty set
4:      while C′ ≠ C do
5:          Set C′ = C
6:          for each pair of pseudomonomials f, g ∈ C do
7:              If f and g are compatible, compute h = f ⊕ g
8:                  Add h to C
9:          Set C := RemoveRedundancy(C)
10:     Return C
```

The remainder of this section will be devoted to proving the correctness of this algorithm. To prove this we introduce the concept of "distance" between a generating set and the canonical form.

Definition 2.75. Let J_C be a neural ideal generated by a set of pseudomonomials $C = \{f_1, \dots f_k\}$. The *distance* of C in J_C is the number of pseudomonomials in J_C that are not multiples of any f_i, and is denoted $D(C)$.

Notice that $D(C)$ is always finite since there are only finitely many pseudomonomials, and also that $D(C) = 0$ if and only if $\mathrm{CF}(J_C) \subseteq C$. In particular, the only redundancy free set of pseudomonomials with $D(C) = 0$ is the canonical form itself. To prove the correctness of Algorithm 2 we will show that $D(C)$ strictly decreases during each iteration of the outer while loop.

Proof of correctness of Algorithm 2. For convience, let J_C be the neural ideal generated by the pseudomonomials in C. Observe that removing redundancy from a set of pseudomonomials does not change its distance. Then let $d = D(C)$ be the distance of C at the beginning of an iteration of the while loop in Algorithm 2. If $d = 0$ then at the end of the loop we will obtain a redundancy free generating set of distance zero, which must be the canonical form. At this point further iterations will not change C since these operations include only the addition of pseudomonomials to C and their subsequent removal due to redundancy. Hence we will have $C = C'$ and break out of the loop, returning C, which will be the canonical form since it is a generating set with distance zero and no redundancy.

If $d > 0$ then we hope to show that $D(C)$ strictly decreases as a result of the operations performed in the loop. Since $d > 0$ there is a nonzero number of pseudomonomials in J_C which are not multiples of elements of C. Among these, choose a pseudomonomial h which is maximal with respect to division. That is, h is not a multiple of any element of C, but any multiple of it will be. Notice that h cannot be of degree n, since $\mathrm{can}(C)$ must be all of the degree n pseudomonomials in J_C (cf. Lemma 2.19 and related discussion in Section 2.2). Thus there must exist some x_i so that neither x_i nor $\overline{x}_i$ is a factor of h.

Now notice that $x_i h$ and $\overline{x}_i h$ are each pseudomonomials, and must be multiples of pseudomonomials in C, say f and g respectively. Observe that we cannot have $f = g$ since then h would be a multiple of $f \in C$. Also note that x_i divides f and $\overline{x}_i$ divides g, but both must agree on all other factors, namely those in f. That is, f and g are compatible pseudomonomials in C. Hence during the while loop in Algorithm 2 we will compute $f \oplus g$, which is a divisor of h since $f \oplus g$ does not contain a factor depending on x_i and agrees with h on all factors present. With $f \oplus g$ in C the pseudomonomials h will no longer contribute to $D(C)$ and so $D(C)$ strictly decreases by at least 1 for every iteration of the while loop.

This implies that we eventually reach a situation in which $D(C) = 0$, which we have already argued results in returning the canonical form. Also note that we cannot break out of the loop sooner than when $D(C) = 0$ since

if $d > 0$ we change C during the loop so it is not equal to C'. This proves that the algorithm terminates and returns the canonical form of $J_C = \langle C \rangle$. □

2.6 Summary

In this chapter our goal has been to understand codes by associating algebraic structures to them. The neural ideal of a code, along with its associated canonical form, were our primary objects of concern. We saw that these two objects can be associated uniquely to any code, and that their structure reveals important information about the code and its potential realizations.

In Section 2.4 we examined a correspondence between transformations of codes and algebraic maps between polynomial rings. We began by characterizing all homomorphisms $\phi : \mathbb{F}_2[n] \to \mathbb{F}_2[m]$ which respect (or preserve) neural ideals. We then broadened our perspective by considering homomorphisms modulo the Boolean relations. We found that such homomorphisms were slightly more general, since we could map multiple variables to the same variable. As described in Theorem 2.3, we are always able to write these homomorphisms as a composition of three fundamental types of maps: indentification, bit flipping, and restriction. To relate these algebraic maps to codes, we described a correspondence between the fundamental maps and transformations of codes. We found that each type of map induced some transformation of codes, and that the link was a special case of these transformations.

We also considered an extremely general class algebraic transformations: $\mathbb{F}_2$-linear transformations from $R[n]$ to $R[m]$. We discovered that these transformations can in fact preserve neural ideals, and even canonical forms. Somewhat surprisingly, the $\mathbb{F}_2$-linear maps preserving canonical forms necessarily arise from hypercube homomorphisms.

Finally, we described how homomorphisms respecting neural ideals affect the canonical form. We showed that when we are not working modulo the Boolean relations, the canonical form remains relatively unaffected under such homomorphisms: applying one of these homomorphisms yields a set of pseudomonomials which is at worst redundant with respect to divisibility. When working mod the Boolean relations things are much less straightforward, and to ameliorate this we described an algorithm for computing the canonical form of a neural ideal from any generating set of pseudomonomials.

This chapter has described a number of algebraic approaches to understanding codes. In the following chapters we will focus on classifying when codes have certain realizations, particularly realizations consisting of convex open sets. To approach this problem we will employ the algebraic tools we have developed here, as well as more purely geometric constructions.

Chapter 3

Geometric Effects of Code Transformations

In the previous chapter we examined algebraic transformations of neural ideals and the effects of these transformations on codes. In this chapter we will attempt to translate these results into the geometric world of realizations. Recall that a realization of a code C is a collection of sets $\mathcal{U}$ in a topological space X so that $C(\mathcal{U}) = C$ (cf. Definition 1.4).

3.1 The Geometric Action of Code Transformations

We found previously that every transformation of a code induced by a homomorphism respecting neural ideals could be expressed by composing maps of three fundamental types:

- Identification (Definition 2.52)
- Bit flipping (Definition 2.56)
- Restriction (Definition 2.58)

Recall that permutations are a special case of identification maps. Given the natural action of these algebraic maps on codes, we aim to take a further step and translate these transformations of codes into transformations of their associated realizations. In particular, if we are given a code transformation $\mathcal{T}$, a code C, and a realization $\mathcal{U} = \{U_1, \ldots, U_n\}$ of C, we seek to modify $\mathcal{U}$ to obtain a realization of $\mathcal{T}(C)$. Our hope is that in modifying $\mathcal{U}$ we can maintain certain properties of the sets in $\mathcal{U}$ such as convexity

and openness. This is not always the case, but we will see that in some important cases these properties are maintained. In this section we will not consider identification maps in their full generality, since they do not have a natural geometric interpretation. Instead we characterize the relatively straightforward special case of permutations.

Lemma 3.1. *Let C be a code with a realization $\mathcal{U} = \{U_1, \ldots, U_n\}$ and let λ be a permutation of $[n]$. Then the set $\mathcal{V} = \{U_{\lambda(1)}, \ldots, U_{\lambda(n)}\}$ is a realization of $\lambda(C)$.*

Proof. It suffices to show that $\mathcal{U}(v)$ is nonempty precisely when $\mathcal{V}(\lambda(v))$ is nonempty for all $v \in \{0,1\}^n$. We will show in fact that $\mathcal{U}(v) = \mathcal{V}(\lambda(v))$. To see this notice that

$$\begin{aligned}\mathcal{U}(v) &= \bigcap_{v_i=1} U_i \setminus \bigcup_{v_j=0} U_j \\ &= \bigcap_{\lambda(v)_i=1} U_{\lambda(i)} \setminus \bigcup_{\lambda(v)_j=0} U_{\lambda(j)} \\ &= \mathcal{V}(\lambda(v)).\end{aligned}$$

Thus the codes of $\mathcal{U}$ and $\mathcal{V}$ are the same up to applying the permutation λ, as desired. □

Corollary 3.2. *Let C be a code and λ be a permutation of $[n]$. Then C is convex if and only if $\lambda(C)$ is convex.*

Proof. This is immediate since permuting the indices of a collection of sets does not affect the convexity of those sets. □

Next we characterize bit flipping maps. Bit flipping maps function as their name suggests, by "flipping" the i-th bit in each codeword from 0 to 1 or vice versa. Their geometric action is to complement the set U_i.

Lemma 3.3. *Let C be a code with a realization $\mathcal{U} = \{U_1, \ldots, U_n\}$ in a space X. Then the collection of sets*

$$\mathcal{V} = \{U_1, \ldots, (X \setminus U_i), \ldots, U_n\}$$

is a realization of $\delta_i(C)$.

Proof. We must show that the codeword regions in $\mathcal{V}$ that are nonempty are precisely those corresponding to elements of $\delta_i(C)$. To do this we write

each vector in $\{0,1\}^n$ as $v \oplus v^i$ and consider $V(v \oplus v^i)$. We consider two cases: either $v_i = 0$ or $v_i = 1$. If $v_i = 0$ then we can write $\mathcal{V}(v \oplus v^i)$ as

$$\begin{aligned}\mathcal{V}(v \oplus v_i) &= \left(\bigcap_{v_j=1} U_j \setminus \bigcup_{v_k=0, k\neq i} U_k\right) \cap (X \setminus U_i) \\ &= \bigcap_{v_j=1} U_j \setminus \bigcup_{v_k=0} U_k \\ &= \mathcal{U}(v).\end{aligned}$$

Similarly, when $v_i = 1$ we have that the i-th bit of $v \oplus v_i$ is 0, and so

$$\begin{aligned}\mathcal{V}(v \oplus v_i) &= \left(\bigcap_{v_j=1, j\neq i} U_j \setminus \bigcup_{v_k=0} U_k\right) \setminus (X \setminus U_i) \\ &= \bigcap_{v_j=1} U_j \setminus \bigcup_{v_k=0, k\neq i} U_k \\ &= \mathcal{U}(v).\end{aligned}$$

In both cases we see that $\mathcal{V}(v \oplus v^i) = \mathcal{U}(v)$ and so $\mathcal{V}(v \oplus v^i)$ is nonempty precisely when $v \in \mathcal{C}$. Thus $\mathcal{C}(\mathcal{V}) = \delta_i(\mathcal{C})$ which proves the desired result. □

Example 3.4. Figure 3.1 shows a convex code $\mathcal{C}$ on 3 neurons, along with a realization of it in $\mathbb{R}^2$. To its right we have the code $\delta_2(\mathcal{C})$, with a (non-convex) realization corresponding to taking the complement of U_2. Clearly flipping bits need not preserve the convexity of a code in general, since taking the complement of a set does not preserve convexity (or even openness).

Definition 3.5. Let $\mathcal{U} = \{U_1, \ldots, U_n\}$ be a collection of sets and let σ and τ be disjoint subsets of $[n]$. The *compatible region* of (σ, τ) in $\mathcal{U}$ is the set

$$\left(\bigcap_{i\in\sigma} U_i\right) \setminus \left(\bigcup_{j\in\tau} U_j\right).$$

The notion of a compatible region generalizes that of a codeword region: it simply includes the points where a particular set of neurons fires while another set is inactive. Note in particular that whenever $\sigma \cup \tau = [n]$ the compatible region is just the codeword region for the vector whose support is σ.

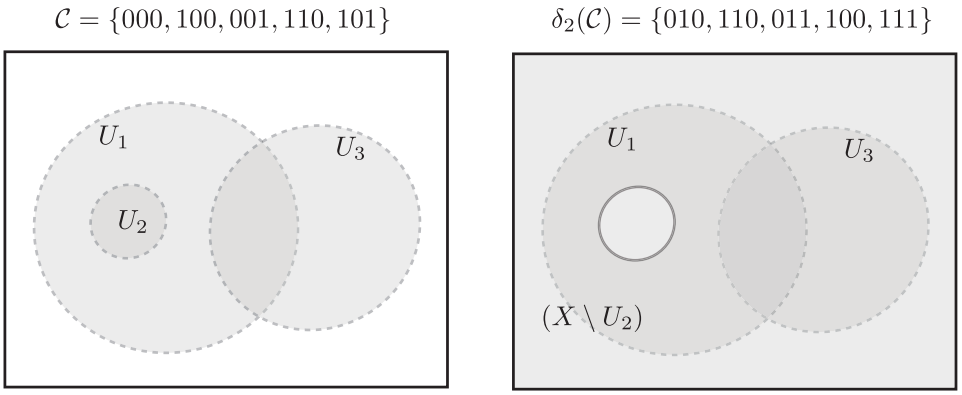


Figure 3.1 A code, its image under the bit flipping map δ_2, and realizations of each in $\mathbb{R}^2$.

Lemma 3.6. *Let C be a code with a realization $\mathcal{U} = \{U_1, \ldots, U_n\}$ in a space X and let m and m' be integers so that $1 \leq m \leq m' \leq n$. Let $\tau = \{m+1, m+2, \ldots, m'\}$ and $\sigma = \{m'+1, m'+2, \ldots, n\}$ and let X' be the compatible region of (σ, τ) in $\mathcal{U}$. Then the collection of sets*

$$\mathcal{V} = \{U_i \cap X' \mid i \in [m]\}$$

is a realization of $\mathrm{rest}(C, m, m')$ *in the toplogical space X'.*

Proof. Let $\gamma \subseteq [m]$ be any vector (as defined by its support) and for each $i \in [m]$ let $V_i = U_i \cap X'$ be the i-th set in $\mathcal{V}$ as described above. We aim to show that $\mathcal{V}(\gamma)$ is nonempty exactly when $\gamma \in \mathrm{rest}(C, m, m')$. To prove this we compute directly that

$$\mathcal{V}(\gamma) = \bigcap_{i \in \gamma} V_i \setminus \bigcup_{j \in [m] \setminus \gamma} V_j \qquad \text{Definition of codeword region } \mathcal{V}(\gamma).$$

$$= \bigcap_{i \in \gamma} (U_i \cap X') \setminus \bigcup_{j \in [m] \setminus \gamma} (U_j \cap X') \qquad \text{Definition of } V_i.$$

$$= X' \cap \left(\bigcap_{i \in \gamma} U_i \setminus \bigcup_{j \in [m] \setminus \gamma} U_j \right) \qquad (A \cap X') \setminus (B \cap X') = X' \cap (A \setminus B).$$

$$= \left(\bigcap_{k \in \sigma} U_k \setminus \bigcup_{l \in \tau} U_l \right) \cap \left(\bigcap_{i \in \gamma} U_i \setminus \bigcup_{j \in [m] \setminus \gamma} U_j \right) \quad X' \text{ is the compatible region of } (\sigma, \tau).$$

$$= \bigcap_{k \in \sigma \cup \gamma} U_k \setminus \bigcup_{l \in \tau \cup ([m] \setminus \gamma)} U_l \quad (A \setminus B) \cap (C \setminus D) = (A \cap C) \setminus (B \cup D).$$

$$= \bigcap_{k \in \sigma \cup \gamma} U_k \setminus \bigcup_{l \in [n] \setminus (\sigma \cup \gamma)} U_l \quad \tau \cup ([m] \setminus \gamma) = [n] \setminus (\sigma \cup \gamma).$$

$$= \mathcal{U}(\sigma \cup \gamma). \quad \text{Definition of codeword region } \mathcal{U}(\sigma \cup \gamma).$$

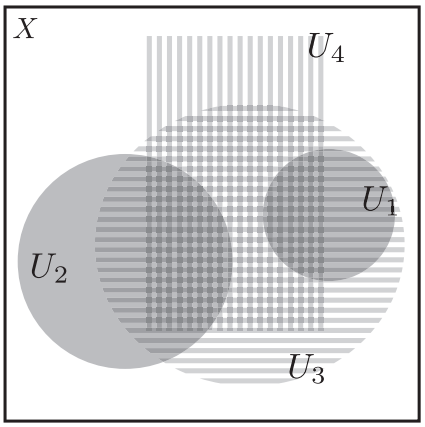
Thus $\gamma \in C(\mathcal{V})$ if and only if $\sigma \cup \gamma$ is in $C(\mathcal{U}) = C$. But $\gamma \in \text{rest}(C, m, m')$ under precisely the same conditions by Definition 2.58. Thus $C(\mathcal{V}) = \text{rest}(C, m, m')$, proving the desired result.

□

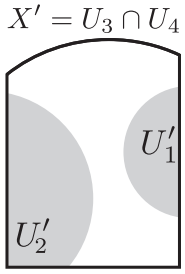
Corollary 3.7. *Let $C \subseteq \mathbb{F}_2^n$ be a code with a convex realization in $\mathbb{R}^d$. Then $\text{Lk}_\sigma(C)$ is a convex code for any $\sigma \subseteq [n]$ and has a realization in $\mathbb{R}^d$.*

Proof. Let $\mathcal{U}$ be a convex realization of C. Recall from Lemma 2.62 that, up to permutation, the link of a code is just a special case of restriction with $m = m'$. If we apply the lemma proven above then we see that the compatible region X will be a convex open set, since it is the intersection of convex open sets. Likewise, each V_i is a convex open set, and so we obtain a realization of $\text{Lk}_\sigma(C)$ consisting of convex open sets in a convex subspace of $\mathbb{R}^d$. Thus $\text{Lk}_\sigma(C)$ is always convex when C is convex. □

Example 3.8. The following example shows how convexity is preserved by linking, as in Corollary 3.7. Taking a link is the same as looking inside a convex "window" in a realization determined by the intersection of sets in the realization. In the example below this "window" is the intersection $U_3 \cap U_4$, which becomes the space for our realization of the link. In the figure U_3 and U_4 are striped to make the sets easier to discern.



$$\mathcal{C} = \left\{ \begin{matrix} 0000, 0100, 0010, 0001, 1010, \\ 0110, 0011, 1101, 0111 \end{matrix} \right\}$$



$$\mathrm{Lk}_{\{3,4\}}(\mathcal{C}) = \mathrm{rest}(\mathcal{C}, 2, 2) = \{00, 10, 01\}$$

3.2 Obstructions to Convexity

The following lemma generalizes the idea of a "local obstruction" presented in Curto et al. (2015). Whereas Curto et al. (2015) considers links only in the simplicial complex of a code, we can also consider links more generally, as described in Definition 1.14. However, this result is also slightly more difficult to work with practically, since it does not reduce the problem of determining convexity to simplicial homology.

Lemma 3.9. *Let C be a code. If there exists $\sigma \subseteq [n]$ so that $\mathrm{Lk}_\sigma(C)$ is not convex in $\mathbb{R}^d$, then C is not convex in $\mathbb{R}^d$.*

Proof. This is the contrapositive of Corollary 3.7. □

In order to work with this result practically, we hope that the codes $\mathrm{Lk}_\sigma(C)$ will be simpler than C itself. Indeed, if σ is nonempty then the link of σ is simpler in the following two ways.

Proposition 3.10. *Let C be a k-sparse code and let $\sigma \subseteq [n]$. Then $\mathrm{Lk}_\sigma(C)$ is a code on no more $n - |\sigma|$ bits and has sparsity no more than $k - |\sigma|$.*

Proof. Since every set in $\mathrm{Lk}_\sigma(C)$ is disjoint from σ it is clear that the link has no more than $n - |\sigma|$ bits which take nonzero values. Thus the link can be thought of as a code on $n - |\sigma|$ bits. Similarly, no set in the link can have

more than $n - |\sigma|$ elements since its union with σ must be a set in C, which is k-sparse. □

The above proposition immediately suggests an algorithm for testing convexity: if we are given a code C then we can recursively take links in the code to obtain codes which are sparse enough to analyze for convexity easily. In Jeffs et al. (2015) the authors characterize the convexity of all 2-sparse codes, and this result could be used by the algorithm to detect some limited obstructions to convexity in codes. However, no such algorithm will be able to detect all obstructions to convexity. The authors in Lienkaemper et al. (2015) describe a code C which is not convex in any dimension, but for which $\mathrm{Lk}_\sigma(C)$ is convex for all σ. Nevertheless, such an algorithmic approach is useful, especially given the connections we have drawn between the link and algebraic transformations of neural ideals.

Chapter 4

Geometric Constructions

In this chapter we approach the problem of convex codes from a purely geometric perspective. In previous chapters we worked with algebraic objects associated to codes such as the neural ideal, but here we diverge from this perspective by working directly with the realizations associated to a code.

4.1 Path connected realizations

Recall from Proposition 1.5 that all codes have a realization in some space. To address this, and to better answer the biological problem of stimulus reconstruction, we generally restrict ourselves to realizations which consist of convex open sets. However, there are many other sensible restrictions to place on sets in a realization. In this section we discuss some possible alternatives to convexity. A much weaker condition than convexity is path connectivity.

Definition 4.1. A set $U \subseteq \mathbb{R}^d$ is *path connected* if for every two points $p, q \in U$ there is a path between p and q which stays in U. More precisely, if for every $p, q \in U$ there exists a continuous map $f : [0,1] \to U$ so that $f(0) = p$ and $f(1) = q$.

Path connected realizations are natural to consider since neural codes arise from spatial stimulus occuring as an animal walks along a path through its environment. In this section we will completely characterize when a code has a realization consisting of path connected open sets. Throughout we will adopt the simplifying assumption that all codes contain

the codeword $00\cdots0$. Recall that the presence of this codeword indicates an "outside space" in the environment where no neurons fire.

Our primary tool in characterizing these codes will be the poset of a code and the link. The poset associated to a code is simply the collection of its codewords ordered with respect to inclusion. Recall from Definition 1.14 that the link of some $\sigma \subseteq [n]$ in a code C is the result of collecting all sets in C containing σ and then removing σ from all these sets. Our result for this section is the following:

Theorem 4.1. *Let C be a code. Then C has a realization consisting of open path connected sets in $\mathbb{R}^3$ if and only if the underlying graph of the poset of $\mathrm{Lk}_{\{i\}}(C)$ is connected for all i.*

We establish this result with the following lemmas. In Lemma 4.4 we will see a slight generalization of the above theorem. In particular, we will see that if C has a realization consisting of open path connected sets and the underlying graph of its poset is planar, then C has a realization in $\mathbb{R}^2$ consisting of path connected open sets. In the lemma below, recall that U_σ is the intersection of all U_i with $i \in \sigma$. Also recall that our convention is to have the empty intersection be the entire space X in which we are attempting to realize the code.

Lemma 4.2. *Let C be a code with a realization $\mathcal{U} = \{U_1, \ldots, U_n\}$ consisting of open sets in a space $X \subseteq \mathbb{R}^d$. Let Σ_i be the collection of minimal elements in the poset of $\mathrm{Lk}_{\{i\}}(C)$ for some index i. Then the set*

$$\Sigma_i(\mathcal{U}) := \{U_i \cap U_\tau | \tau \in \Sigma_i\}$$

is an open cover of U_i.

Proof. Recall that U_i is the union of all codeword regions $\mathcal{U}(\sigma)$ with $\sigma \in C$ and $i \in \sigma$. We know that $\mathcal{U}(\sigma) \subseteq U_\sigma$ since $\mathcal{U}(\sigma)$ is obtained from U_σ by potentially removing points. Also observe that $U_\sigma \subseteq U_\tau$ if and only if $\tau \subseteq \sigma$. Finally, we know from the definition of the link that $\tau \in \mathrm{Lk}_{\{i\}}(C)$ if and only if $\tau \cup \{i\} \in C$ and $i \notin \tau$. With this information we can compute straightforwardly that

$$\begin{aligned} U_i &= \bigcup_{i \in \sigma \in C} \mathcal{U}(\sigma) \\ &\subseteq \bigcup_{i \in \sigma \in C} U_\sigma \qquad\qquad \text{Since } \mathcal{U}(\sigma) \subseteq U_\sigma. \end{aligned}$$

$$
\begin{aligned}
&= \bigcup_{\tau \in \mathrm{Lk}_{\{i\}}(C)} U_{\{i\}\cup\tau} && \text{By definition of the link.}\\
&\subseteq \bigcup_{\tau \in \Sigma_i} U_{\{i\}\cup\tau} && \text{Since } U_\tau \subseteq U_{\tau'} \Leftrightarrow \tau' \subseteq \tau.\\
&= \bigcup_{\tau \in \Sigma_i} U_i \cap U_\tau\\
&= \bigcup_{U \in \Sigma_i(\mathcal{U})} U.
\end{aligned}
$$

Thus the collection of sets $\Sigma_i(\mathcal{U})$ covers U_i, as desired. Clearly each set in the collection is open since it is the intersection of open sets. □

Lemma 4.3. *Let C be a code with a realization consisting of path connected open sets. Then the underlying graph of the poset of* $\mathrm{Lk}_{\{i\}}(C)$ *is connected for all i.*

Proof. We proceed by contrapositive. Let $\mathcal{U} = \{U_1, \ldots, U_n\}$ be any realization of C consisting of open sets, and suppose without loss of generality that the underlying graph of the poset of $\mathrm{Lk}_{\{1\}}(C)$ is not connected. Let $\Sigma_1^{(1)}, \Sigma_1^{(2)}, \ldots, \Sigma_1^{(k)}$ be the collection of minimal elements in each connected component of the underlying graph of the poset of $\mathrm{Lk}_{\{1\}}(C)$. Notice that the various $\Sigma_1^{(i)}$ partition Σ_1. Also notice that none of these can contain the empty set, since otherwise the poset would have a unique minimal element and be connected.

For each $i \in [k]$ define

$$V_i = \bigcup_{\tau \in \Sigma_1^{(i)}} U_1 \cap U_\tau.$$

Notice that each V_i is a union of sets in $\Sigma_1(\mathcal{U})$ since each set is $U_1 \cap U_\tau$ for some minimal element τ of the poset of $\mathrm{Lk}_{\{1\}}(C)$. Moreover, any set $U_1 \cap U_\tau$ in $\Sigma_1(\mathcal{U})$ is contained in some V_i, namely the V_i for which $\tau \in \Sigma_1^{(i)}$. Thus the collection of V_i forms an open cover of U_1.

We now aim to show that V_i and V_j are disjoint whenever $i \neq j$. Suppose that there were some point of intersection between V_i and V_j, corresponding to a codeword $\gamma \in C$. Then γ must contain some $\tau_i \in \Sigma_1^{(i)}$ and some $\tau_j \in \Sigma_1^{(j)}$. But clearly τ_i and τ_j are in the same connected component of $\mathrm{Lk}_{\{1\}}(C)$ since there is a path from each of them to γ. This implies that $i = j$, and in particular V_i and V_j are disjoint whenever $i \neq j$.

Now since we have at least two connected components in $\mathrm{Lk}_{\{1\}}(C)$ we have at least two distinct V_i. Thus the collection of V_i is an open cover of U_1 consisting of disjoint sets, and we conclude that U_1 is not connected. Thus C has a realization consisting of path connected open sets only if the underlying graph of the poset of $\mathrm{Lk}_{\{i\}}(C)$ is connected for all i. □

Lemma 4.4. *Let C be a code and suppose that the underlying graph of the poset of $\mathrm{Lk}_{\{i\}}(C)$ is connected for all i. Let G be the underlying graph of the poset of $C \setminus \{\emptyset\}$. If G has an embedding in $\mathbb{R}^d$ then C has a realization in $\mathbb{R}^d$ consisting of path connected open sets.*

The proof below is constructive but also heavy in notation. For a concrete example of the construction given in the proof one can skip ahead to Example 4.5.

Proof. Consider an embedding of the graph G in $\mathbb{R}^d$. Let $\epsilon_1 > 0$ be sufficiently small so that no two vertices in the embedding are within $3\epsilon_1$ distance of one another, and no line in the embedding passes within $2\epsilon_1$ distance of a vertex unless it leads to that vertex. Then for each $\sigma \in C \setminus \{\emptyset\}$ let B_σ be an open ball of radius ϵ around the vertex corresponding to the codeword σ. By construction none of these balls intersect, nor do they fully cover any line between vertices in the embedding.

For any line between vertices σ and σ', define $l_\sigma^{\sigma'}$ to be the collection of points on the line that are not in B_σ or $B_{\sigma'}$. To keep this labelling consistent we will always assume $\sigma \subset \sigma'$. Notice that none of these are empty since the open balls B_σ do not cover any line fully. Also notice that all of the $l_\sigma^{\sigma'}$ are disjoint, closed, bounded sets. Thus all of these sets have positive distance between them pairwise. Let ϵ_2 be one third of the minimum of these distances. We then associate each of these line segments to an open set containing it. In particular, we define

$$L_\sigma^{\sigma'} = \bigcup_{p \in l_\sigma^{\sigma'}} N(p, \epsilon_2)$$

where $N(p, \epsilon_2)$ is the open ball of radius ϵ_2 centered at p. Clearly each of these sets is open, contains $l_\sigma^{\sigma'}$, and no two of them intersect by our choice of ϵ_2.

To construct our realization we define

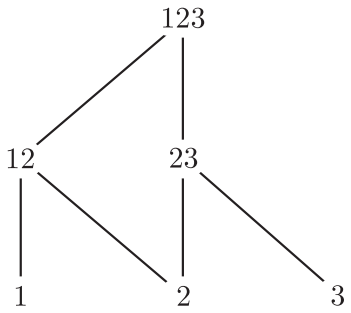
$$U_i = \left(\bigcup_{i \in \sigma} B_\sigma \right) \cup \left(\bigcup_{i \in \sigma \subset \sigma'} L_\sigma^{\sigma'} \right).$$

We claim that the collection $\mathcal{U} = \{U_1, \ldots, U_n\}$ is a realization of C consisting of path connected open sets. Each U_i is open because it is the union of open sets. Each U_i is path connected since $\mathrm{Lk}_{\{i\}}(C)$ is connected for all i. In particular, U_i consists of path connected sets (the B_σ and $L_\sigma^{\sigma'}$) which together cover the embedding of the graph of $\mathrm{Lk}_{\{i\}}(C)$. Since this embedding is connected this cover itself is connected.

It remains to show that $\mathcal{U}$ is a realization of C. First notice that $C(\mathcal{U})$ contains the empty set since the various U_i do not cover all of $\mathbb{R}^d$. Then let σ be any nonempty codeword in C. The set B_σ will yield σ in $C(\mathcal{U})$ since, by construction, $B_\sigma \subseteq U_i$ if and only if $i \in \sigma$. Finally, let $\gamma \subseteq [n]$ be any set which is not in C. We must show that $\gamma \notin C(\mathcal{U})$. In particular, we want to show that $\mathcal{U}(\gamma) = \emptyset$. To do this it suffices to show that no B_σ or $L_\sigma^{\sigma'}$ contains points in $\mathcal{U}(\gamma)$. First notice that each $B_\sigma \subseteq \mathcal{U}(\sigma)$ and since $\gamma \notin C$ each B_σ is disjoint from $\mathcal{U}(\gamma)$. Furthermore, points in $L_\sigma^{\sigma'}$ that are not in $B_{\sigma'}$ will be in $\mathcal{U}(\sigma)$, since these points are covered precisely by the U_i for which $i \in \sigma$. Again since $\gamma \notin C$ we have that these points are not in $\mathcal{U}(\gamma)$. Thus $\mathcal{U}(\gamma)$ is disjoint from all U_i, and is therefore empty. This proves that $\gamma \notin C(\mathcal{U})$ as desired. □

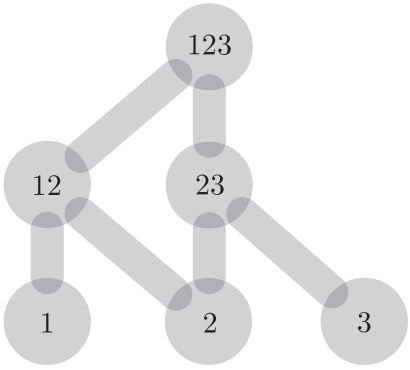
In the example below we use the construction described in this proof to create a path connected realization of a code in $\mathbb{R}^2$.

Example 4.5. Consider the code $C = 2^{[3]} \setminus \{13\} = \{\emptyset, 1, 2, 3, 12, 13, 123\}$. The underlying graph of the poset of $C \setminus \emptyset$ is planar. In particular, the following is an embedding of this graph:



We have drawn the embedding using straight lines, but one need not do so in general. Proceding as in the proof of Lemma 4.4 we construct open

balls around each vertex in this graph. We then take the line segments not covered by these balls and inflate them slightly to obtain open sets. This gives us the following collection of sets:



We have left off the labels B_σ and $L_\sigma^{\sigma'}$ described in the proof of Lemma 4.4 in order to avoid clutter. To finish constructing our realization, we form each U_i by taking the union of the lines and balls which involve the index i. This yields a realization of C in $\mathbb{R}^2$ using open path connected sets.

In the figure above each set U_1, U_2, and U_3 is patterned with lines in a particular direction. Regions in the figure where there are grids correspond to overlap between the sets.

With these tools developed we can return to our overall result.

Proof of Theorem 4.1. The forward direction follows immediately from Lemma 4.3. In particular, the assumption that C has a realization consisting of path connected open sets in $\mathbb{R}^3$ is stronger than the hypothesis in Lemma 4.3, which

places no restrictions on the space in which C has a path connected realization.

For the reverse direction, we use the fact that any graph has an embedding in $\mathbb{R}^3$. In particular, the underlying graph of the poset of $C \setminus \{\emptyset\}$ has an embedding in $\mathbb{R}^3$. Furthermore, we have by hypothesis that $\mathrm{Lk}_{\{i\}}(C)$ is connected for all i, and so by Lemma 4.4 we know that C has a path connected realization in $\mathbb{R}^3$. □

This theorem provides a bound on the dimension necessary to realize a code using path connected sets which does not depend on the code. We know that if we can realize a code then we can always do so in $\mathbb{R}^3$, and it is natural to ask when we can go to $\mathbb{R}^2$ or even $\mathbb{R}^1$. The bound of $\mathbb{R}^3$ is sharp in the sense that there exist codes which have path connected realizations in $\mathbb{R}^3$ and not $\mathbb{R}^2$. One class of counterexamples can be obtained by subdividing planar graphs, as in Jeffs et al. (2015). As of yet there is no characterization of which codes have path connected realizations in $\mathbb{R}^3$ but not $\mathbb{R}^2$.

In the construction described in Lemma 4.4 the sets we obtain are not necessarily simply connected. For instance in Example 4.5 the set U_2 has nontrivial fundamental group. Simply connected sets are a reasonable intermediary between path connected sets and convex sets, since convex sets are always simply connected. A potentially interesting question is whether codes with path connected realizations will always have simply connected realizations. Investigating which codes do have realizations consisting of simply connected sets may also have the potential to shed light on which codes have convex realizations.

Chapter 5

Conclusion

We have approached neural codes from several angles. First, we examined existing algebraic objects such as the neural ideal and the canonical form in Chapter 2. We saw that, as described in Curto et al. (2013), these objects yield useful information about the code and its realizations. In particular the canonical form compactly encodes minimal relationships among sets in any realization of the associated code. Given the useful nature of these objects, we sought to understand them better by examining their algebraic structure. We completely classified all homomorphisms that map neural ideals to neural ideals, and we also briefly investigated $\mathbb{F}_2$-linear transformations with the same properties. Our classification yielded a decomposition of any homomorphism respecting neural ideals into a composition of four natural types of homomorphisms.

This classification also gave us a natural way to describe how these homomorphisms act on the underlying codes. In Section 2.4.4 we found that, up to composition, we can describe the effects on codes in terms of permuting indices, flipping bits, restricting to sets of "compatible" codewords, and identification maps. The last of these turns out to be valid only when we work modulo the Boolean relations, and it is in some sense it is less natural than the other classes of maps.

We also investigated whether these homomorphisms could be used to compute the canonical forms of related codes easily. We found that homomorphisms $\phi : \mathbb{F}_2[n] \to \mathbb{F}_2[m]$ that respect neural ideals do have some use: one can obtain the canonical form of $\phi(J_\mathcal{C})$ by applying ϕ to the canonical form of $J_\mathcal{C}$ and removing pseudomonomials that are redundant to one another. This method of computing the canonical form is somewhat simpler than computing the canonical form directly from a code. We saw that this

relationship does not hold when we work modulo the Boolean relations. Finally, we described an algorithm for computing the canonical form from any generating set for the neural ideal consisting of pseudomonomials.

In Chapter 3 we sought to understand transformations of codes geometrically. We characterized certain relationships between realizations of codes and realizations of transformed codes. Importantly, we showed that some code transformations, such as taking a link, preserve the convexity of a code. Since these transformations may reduce the complexity of a code, they can serve as a coarse test for nonconvexity: given a code, we simplify it and then test for nonconvexity. If the resulting code is not convex, then the original code could not have been convex. Unfortunately, these tests are rarely complete since they do not detect all nonconvex codes. Nevertheless, when coupled with the algebraic understanding developed in Chapter 2, these tests can serve as an efficient first check for nonconvexity. Conceptually, the content of Chapter 3 is a direct bridge between realizations of codes and the algebraic world of their neural ideals. Previous work has shown correspondences of a similar style: for example Curto et al. (2013) described how relationships between sets in realizations appear in the canonical form. The work in Chapter 3 is novel in that it focuses on transformations of the objects of concern, namely neural ideals and realizations of their underlying codes. The direct correspondence between these transformations is somewhat surprising, since there is no reason a priori to expect that geometric operations would be associated to with algebraic transformations, or vice versa.

In Chapter 4 we took a concrete geometric approach, providing a characterization of codes which have realizations consisting of path connected open sets. We noticed that these realizations can always be achieved in $\mathbb{R}^3$, but that bounding the dimension more precisely was a difficult task. We also briefly considered simply connected realizations as an intermediary between path connected sets and convex sets.

Bibliography

Curto, Carina, Elizabeth Gross, Jack Jeffries, Katherine Morrison, Mohamed Omar, Zvi Rosen, Anne Shiu, and Nora Youngs. 2015. What makes a neural code convex? 1508.00150.

Curto, Carina, and Vladimir Itskov. 2008. Cell groups reveal structure of stimulus space. *PLoS Computational Biology* .

Curto, Carina, Vladimir Itskov, Alan Veliz-Cuba, and Nora Youngs. 2013. The neural ring: an algebraic tool for analyzing the intrinsic structure of neural codes 1212.4201.

Curto, Carina, and Nora Youngs. 2015. Neural ring homomorphisms and maps between neural codes 1511.00255.

Jeffs, R. Amzi, Mohamed Omar, Natchanon Suaysom, Aleina Wachtel, and Nora Youngs. 2015. Sparse neural codes and convexity 1511.00283.

Lienkaemper, Caitlin, Anne Shiu, and Zev Woodstock. 2015. Obstructions to convexity in neural codes 1509.03328.

O'Keefe, John, and Jonathan Dostrovsky. 1971. The hippocampus as a spatial map. preliminary evidence from unit activity in the freely-moving rat. *Brain Research* 171–175.