PHONOLOGICAL EXPRESSIVITY AND LEARNING VIA BOOLEAN MONADIC RECURSIVE SCHEMES

By

TATEVIK ARAYEVNA YOLYAN

A dissertation submitted to the

School of Graduate Studies

Rutgers, The State University of New Jersey

In partial fulfillment of the requirements

For the degree of

Doctor of Philosophy

Graduate Program in Linguistics

Written under the direction of

Adam Jardine

And approved by

_____

_____

_____

_____

New Brunswick, New Jersey

October 2025

ABSTRACT OF THE DISSERTATION

Phonological Expressivity and Learning
via Boolean Monadic Recursive Schemes

by TATEVIK ARAYEVNA YOLYAN

Dissertation Director:
Adam Jardine

This dissertation discusses the representation, expressivity, and computational learning of phonological maps through the model-theoretic framework of Boolean Monadic Recursive Schemes (BMRS). The BMRS framework is used to expresses phonological processes as logical transductions over relational structures. This framework has been proposed as an alternative to finite state transducers (FSTs) in computational phonology by Chandlee and Jardine [2021] because it combines the restrictiveness of FSTs with the descriptive capabilities of model theory. This dissertation presents the BMRS framework with an in-depth discussion of its connections to model theory, formal language theory, and linguistics, and showcases the merits of BMRS for computational phonology by revisiting expressivity and learning from a logical perspective.

With respect to expressivity, this dissertation presents a logical characterization of the weakly deterministic class of functions, which were originally proposed by Heinz and Lai [2013] as a description of the expressivity of natural language phonological maps. Although FST characterizations of this class have been proposed in recent years, these characterizations have not been able to decisively distinguish between weakly deterministic and more complex maps. This dissertation shows that a logical characterization does succeed in correctly identifying weakly deterministic maps, and can be used to reason about what maps are outside this region, thus yielding a testable hypothesis about the expressivity of natural language maps.

With respect to learning, this dissertation develops a learning procedure that adapts phonotactic learning with model-theoretic representations to learning transductions. This learning procedure

uses a partially-ordered hypothesis space of feature-based string models, and shows how this structure can be employed to learn phonological generalizations from a small number of data points. The significance of the logical perspective is that the learning procedure can be extended to more complex phonological structures that are otherwise difficult to define FSTs over. This dissertation further shows how this work extends to a procedure for learning non-interacting processes from their composition, which remains an open problem for FSTs.

Together, these contributions advance the research program of model-theoretic phonology by establishing BMRS as a robust theoretical and computational framework for phonology.

ACKNOWLEDGMENTS

In the six years I spent working on this PhD, I have been blessed with amazing professors, advisors, friends, and family. I live my life by making lists. So here's my acknowledgments section, in the best way I know how to approach it.

Ed Keenan - Thank you, first and foremost, for being the reason I became a linguist. A decade ago I took a class called "Mathematical Structures in Language" out of pure curiosity. This class was the starting point for the sequence of events that led to this dissertation. Thank you Ed, for allowing a random undergrad with no linguistics background to sit in on your seminars. Thank you for enthusiastically welcoming me to your class, and guiding me through career planning (and telling me to apply to Rutgers). I don't know where I would be right now if I had not stumbled into your classroom, but I certainly would not be completing a PhD in linguistics. I am fortunate to have had the privilege to learn from you.

Adam J - Thank you for being an awesome advisor. The first time I met you was at the open house for prospective students. I mentioned that I had never taken a phonology class in my life. Your response was a nonchalant "you'll figure it out". Thank you for believing in me from day 0. Thank you for knowing when to provide guidance, and when to back off and give me space to figure things out on my own. Thank you for trusting in my abilities as a researcher and a writer. Thank you for constantly encouraging me to do things I would not have the confidence to do otherwise, like submitting to a journal or applying for a competitive fellowship. Thank you for hyping up my ideas at every meeting. I thoroughly enjoyed writing this dissertation, and I am positive it's because of the genuine excitement you've expressed every step of the way.

Adam M - Thank you for continuously being the voice of affirmation when I need it. I came into this program without a linguistics degree and always felt like an oddball and a fraud. Thank you for making me feel that I belong when I felt certain that somebody had made a mistake in accepting me into the program. Thank you for believing in me, and encouraging me to submit to my first conference - a phonology conference! Thank you (and your wife) for opening up your home on Thanksgiving, and treating students like family.

Bruce - Thank you for serving on my committees and always having difficult and insightful questions and comments about my work. Thank you also for teaching some of the most interesting courses I have ever taken. After years of telling myself "I should learn Prolog", while *The Art of Prolog* collected dust on my bookshelf, I finally got around to it because of your Formal Methods class.

Siddharth - Thank you for agreeing to serve on the committee of a linguistics dissertation. Your summer mini-course at Stony Brook was the inspiration for how I approach the first two chapters of this dissertation. I'm grateful to have learned from you, and to have you be a part of this project.

Troy - Thank you for advising my first qualifying paper, and helping me get through a topic that I had absolutely no business writing about. Thank you for all the care and patience you put into being an advisor. That fact that I survived a syntax QP is a testament to what a fantastic advisor you are because, well, we all know how I feel about syntax.

Kristen and Crystal - Thank you for all the work you did as undergraduate program directors during the years that I was a teaching assistant. Thank you for the wonderful conversations we've had about teaching, and for your constructive feedback and advice. Thank you for believing in my teaching abilities, and providing me opportunities to gain experience as a instructor.

Karma Cat Rescue Society - Thank you for getting me through the COVID lockdown with an endless supply of foster kittens to care for.

Indira - Thank you for six years of friendship that I hope will continue for a lifetime. Thank you for all the big things, like giving me a place to stay when I didn't have an apartment. And for all the little things, like coming to class with a backpack full of ginger candy in case I had nausea. Thank you for the adventures, the conversations, and the hilarious mishaps you've dragged me into. Thank you for being a wonderful roommate, and filling our home with warmth and laughter. Your presence was always felt even when you weren't there (because your Echo Dot played Rasputin on full volume at 7 in the morning).

Jesse - Thank you for all the support that made this accomplishment possible. I wrote this dissertation while pregnant and raising a toddler. I got through it because I had an amazing husband, who went out of his way to give me the space and support I need to reach my goals. Thank you for waking up early to clean, for bringing me coffee in bed, and for taking Amalia out for father-daughter time so that I can work. Thank you for not questioning the absurdity of the little things that make me overwhelmed, and for simply providing unconditional support and love. Thank you for the conversations over a whiteboard that helped develop my ideas. Thank you for making life fun and adventurous, and for keeping calm in the midst of chaos.

Mom - Where do I even start? Or end? Thank you for literally all of it? Thank you for being the most inspiring person I will ever know. Thank you for your strength and resilience. Thank you for being the foundation on which everything else exists.

Amalia and Elias - Thank you for filling every day of my life with joy. Every morning starts with a smile and every night ends with a hug. Thank you for making all the normal and mundane things in life beautiful and exciting.

The Andrew W. Mellon Foundation - Thank you for funding my final year through the Dissertation Completion Fellowship. I am incredibly fortunate to have had the financial support to fully dedicate myself to writing this dissertation.

TABLE OF CONTENTS

1 INTRODUCTION

This dissertation is grounded in formal language theory and the Chomsky Hierarchy of grammars, which provide a formal basis for examining the expressivity of various linguistic processes [Chomsky, 1956, 1959, Jäager and Rogers, 2012, Heinz, 2018]. Phonological grammars, in particular, are argued to be strictly contained within the restrictive class of regular grammars [Johnson, 1972, Kaplan and Kay, 1994]. From a computational perspective, this result means that within natural language phonological patterns, grammatical surface (phonetic) forms can be represented by finite state acceptors and the relationships between underlying and surface forms can be represented by finite state transducers (FSTs) [Hulden, 2009, Beesley and Karttunen, 2003, Mohri, 1997]. This result has lead to generalizations about the expressivity of cross-linguistic phonological patterns [Luo, 2017, Jardine, 2016a, Heinz and Lai, 2013, Gainor et al., 2012], the development of learning algorithms for structured subclasses of regular grammars [Burness and McMullin, 2019, Chandlee et al., 2014, Chandlee, 2014, Jardine et al., 2014, Heinz, 2010b, Oncina et al., 1993], and perspectives on the cognitive complexity of the phonological component of grammar [De Santo and Rawski, 2022, Rogers et al., 2013, Heinz and Idsardi, 2013, 2011]. The regular bound on phonological grammars is a significant starting point for this dissertation because BMRS can be used to represent string functions within this class as *logical* transductions. This dissertation discusses the shift from finite state to logical transducers in computational phonology, and revisits the research themes of representation, expressivity and learning in order to showcase the merits of model theory and BMRS as an alternative to formal language theory in computational phonology.

1.1 Model-Theoretic Representations in Linguistics

The central concept underlying this dissertation is relational structures. A relational structure consists of a domain of objects, predicates which represents properties of the objects in the domain, and functions and relations over objects in the domain. Within linguistics, these mathematical objects are used to formally describe the various structures used for linguistic analysis. Model-theoretic syntax, for example, uses relational structures to formally represent syntactic trees. Model-theoretic approaches in syntax have been pursued by Rogers [1998], King [1989], Pullum and Scholz [2001], Rogers [2003]. Rogers [1998] describes syntactic trees using relational structures in which the objects in the domain are nodes of a tree. These objects have three binary relations defined over them: the parent relation, the dominance relation, and the linear (precedence) ordering relation. These three relations encode the relevant information in a syntactic tree. The dominance relation, for example, encodes constituency structure, while the precedence relation encodes the left-to-right ordering on constituents. Formalizing tree structures as relational structures provides a means of formalizing concepts that make it possible to describe and derive formal properties that are otherwise elusive. As Rogers [1998] explains:

> The value of such formalizations, beyond providing a basis for reasoning formally about the consequences of a theory, is that they frequently raise linguistically significant issues that are obscured in less rigorous expositions. [...] More concretely, formalized principles may in some cases be simpler than the original statements of some of those principles. The identification component of Rizzi's ECP, for example, reduces in our treatment to a simple requirement that every node occur in a well-formed chain. Although such results are typically only theoretically motivated they may well suggest refinements to the original theories that can be justified empirically. [Rogers, 1998, pg.8]

Within model-theoretic phonology, relational structures are used to represent the internal structure of words, which ultimately represent underlying and surface forms in phonological maps. These structures have a precedence relation which encodes the linear ordering of sounds in a word. Moreover, relational structures allow for predicates which can be used to encode the phonological features which hold for each sound in a word [Chandlee and Jardine, 2021, Chandlee et al., 2019]. This makes it possible to represent a word as a sequence of feature matrices. These structures can also be further enhanced with relations that encode autosegmental structure [Jardine, 2017, Chandlee and Jardine, 2019b], syllable structure [Strother-Garcia, 2019, Strother-Garcia and Heinz, 2017], and prosodic structure [Joo and Jardine, 2025, Dolatian, 2020, Dolatian et al., 2021a]. Phonological

maps between the input and output structures are represented by logical transductions which define the output (surface) structure in terms of logical formulas over the input (underlying) structure [Courcelle, 1994, Engelfriet and Hoogeboom, 2001, Filiot, 2015]. Such model-theoretic representations have been used to characterize subregular classes of phonological grammars [Lambert and Rogers, 2020, Rogers and Lambert, 2019, Rogers and Pullum, 2011] and to develop grammatical inference algorithms [Rawski, 2021, Chandlee et al., 2019, Strother-Garcia et al., 2016, Vu et al., 2018]. Further recent applications and discussions of model-theoretic phonology can be found in [Lambert et al., 2021, Rogers et al., 2013, Graf, 2010, Potts and Pullum, 2002].

The framework of BMRS provides an abstract programming language for expressing logical transductions between relational structures [Bhaskar et al., 2020]. While FSTs represent the underlying and surface forms of words as strings of characters with a linear ordering, BMRS is built off model-theoretic representations which make it possible to encode transductions over feature specifications and more complex structures, and consequently represent phonological generalizations. For this reason, Chandlee and Jardine [2021] propose BMRS as an alternative to FSTs:

> A theory that both incorporates computational characterizations and intensionally captures linguistically significant generalizations has so far proved elusive. [...] the feature-based representations prominent throughout phonological theorizing have not been widely pursued in finite-state analyses of phonological patterns with transitions instead being labeled with unanalyzed segments. [...] Logical descriptions, instead, can capture the same generalizations about the complexity of phonological patterns, and can do so with more realistic phonological representations, such as features, syllable structure, or autosegmental representations. [Chandlee and Jardine, 2021, pg.7-8]

1.2 Expressivity and Restrictiveness

The appeal of model-theoretic representations within phonology is that they allow for rich representations that can encode phonological structure and generalizations. A further appeal of the BMRS framework, in particular, is that BMRS programs represent exactly the class of rational string functions [Bhaskar et al., 2023]. The rational functions are characterized as those that can be expressed by (one-way) non-deterministic finite state transducers [Filiot and Reynier, 2016]. The linguistic significance of the rational string functions is summarized by the concepts of expressivity and restrictiveness. With respect to expressivity, the rational functions can be expressed as rewrite rules in Chomsky and Halle [1968]'s *Sound Patterns of English* (SPE) formalism [Johnson, 1972,

Kaplan and Kay, 1994]. With respect to restrictiveness, functions outside the rational class are hypothesized to be unattested in natural language phonology [Heinz, 2018]. Heinz [2018] discusses the significance of expressivity and restrictiveness as follows:

> A good theory must be both *sufficiently expressive* to accurately describe the actual phonologies in the world's languages and *maximally restrictive*. [...] It is easy to find a sufficiently expressive theory of phonology which is not restrictive. The widely held Church-Turing thesis states that anything that can be calculated or computed can be computed by Turing machines[...]. If phonologists believe their theories and models of phonology are computable then there is already a sufficiently expressive theory of phonology available. The problem with this theory is that it is unrestrictive because it says everything that is computable is possible. The mere existence of a phonology will never be sufficient grounds for dismissing the Church-Turing Theory of Phonology. All of this is a way of saying that a theory not only needs to explain what *there is, but also what there is not.* [Heinz, 2018, pg.135-136]

The equivalence between BMRS programs and the rational functions means that BMRS programs are expressive enough to represent phonological maps, without being too powerful to represent maps which are outside of the realm of natural language sound patterns. BMRS is therefore well-suited for building a theory of phonology because it can represent phonological structure, encode phonological generalization, and 'has a well-understood complexity bound that corresponds to previous results in the study of computational phonology' [Chandlee and Jardine, 2021, pg.3].

1.3 Learning

Restrictive theories capture the structural biases that are inherent to natural language by providing a formal boundary on what types of structures and processes are (and are not) compatible with natural language. These structural biases provide insight into how languages are learned quickly and efficiently [Lambert et al., 2021, Heinz, 2016, 2010b]. Restrictiveness is therefore a desirable property of linguistic theories because it captures restrictions within the learning space of natural language grammars. Chomsky [1965] relates restrictiveness and learning as follows:

> A theory of linguistic structure that aims for explanatory adequacy incorporates an account of linguistic universals, and it attributes tacit knowledge of these universals to the child. It proposes, then, that the child approaches the data with the presumption that they are drawn from a language of a certain antecedently well-defined type, his problem being to determine which of the (humanly) possible languages is that of the community in which he is placed. Language learning would be impossible unless this were the case. The important question is: *What are the initial assumptions concerning the nature of language that the child brings to language learning, and how detailed and specific is the innate schema (the general definition*

> *of 'grammar') that gradually becomes more explicit and differentiated as the child learns the language?* [...] Consequently, the main task of linguistic theory must be to develop an account of linguistic universals that, on the one hand, will not be falsified by the actual diversity of languages and, on the other, will be sufficiently rich and explicit to account for the rapidity and uniformity of language learning, and the remarkable complexity and range of the generative grammars that are the product of language learning. [Chomsky, 1965, pg.27-28]

With respect to phonology, the rational boundary on phonological complexity significantly reduces the space of all possible phonological maps to those which are expressible with a one-way FST. In terms of computation, this means that all phonological maps, including those with long-distance dependencies, require only finite memory. This is in contrast to syntactic dependencies, which are outside of the regular boundary [Chomsky, 1959, Shieber, 1985, Joshi, 1985]. Many phonological patterns fall within even smaller subregular regions which are characterized by deterministic FSTs [Luo, 2017, Payne, 2017, Chandlee and Heinz, 2012] and local dependencies [Chandlee and Heinz, 2018, Chandlee et al., 2015, Chandlee, 2014]. Artificial grammar learning experiments have shown that these characterizations of phonological complexity corresponds with human learning [McMullin and Hansson, 2019, Finley, 2017, 2012, 2011, 2008, Lai, 2015, Finley and Badecker, 2008]. Finley [2008], for example, shows that learners are biased against the Majority Rules harmony pattern which is outside the rational boundary. These computational characterizations have also lead to various learning algorithms for phonological patterns, some of which can be found in Chandlee et al. [2014], Jardine et al. [2014], Heinz [2011, 2009], Gildea and Jurafsky [1995], Oncina et al. [1993]. Thus, a restrictive theory of phonology reflects natural language learning biases and enables efficient learning algorithms.

1.4 Contribution and Outline of Dissertation

The three research themes in Sections 1.1-1.3 are interconnected in the following way. Formal representations of linguistic structure allows us to derive computational result about the expressivity and restrictiveness of natural language grammars. These restrictions represent biases in the learning space, which allow us to develop efficient learning algorithms for those grammars. These learning algorithms are built off formal representations, and learn the linguistic generalizations that are encoded in those representations. This dissertation re-examines these themes for phonological maps, through the lens of BMRS. With respect to representation, this dissertation connects the

BMRS framework with recent work in model-theoretic phonology such as Strother-Garcia [2019], Chandlee and Jardine [2019a], and Strother-Garcia and Heinz [2017]. With respect to expressivity and restrictiveness, this work extends discussions of phonological expressivity from Meinhardt et al. [2021], McCollum et al. [2018] and Heinz and Lai [2013], and provides linguistic context for the logical characterizations of subregular string functions from Bhaskar et al. [2020, 2023] and Chandlee and Lindell [forth.]. With respect to learning, this dissertation extends work on model-theoretic learning of phonotactics from Rawski [2021], Chandlee et al. [2019], and Strother-Garcia et al. [2016] to transformations via BMRS transductions. The fundamental contribution of this dissertation is therefore an in-depth discussion of the merits of the BMRS framework for phonological theory.

Chapters 2 and 3 are background chapters which explore the theme of *representation*. Chapter 2 presents the relevant mathematical definitions and concepts. The first part of this chapter defines relational structures, discusses previous work from Strother-Garcia [2019], Chandlee et al. [2019], and Chandlee and Jardine [2021] on adapting relational structures for phonological representations, and introduces the concept of logical transductions. The second part of the chapter presents the BMRS framework, and shows how it can be used to represent phonological maps as quantifier-free logical transductions.

Chapter 3 discusses the rational bound on phonological maps, and the various nested regions of the subregular hierarchy. The first part of the chapter presents cross-linguistic data that motivate and illustrate the various regions of the subregular hierarchy, and discusses the computational properties of these nested regions in terms of FST representations. The second part of the chapter shows how FSTs can be translated to BMRS programs which simulate the same computation. This discussion provides intuition for logical characterizations of string functions given by Bhaskar et al. [2020, 2023] and Chandlee and Lindell [forth.].

Chapters 4 and Chapter 5 explore the themes of *representation and learning*. Chapter 4 shows how the model-theoretic learning procedure for phonotactics, introduced by Chandlee et al. [2019] and Rawski [2021], can be extended to a learning procedure for (local) phonological *maps* via the BMRS framework. In particular, this chapter shows how the hypothesis space for phonotactic constraints can be adapted to a hypothesis space for environments in which features undergo change. Section 4.3.2 then presents a case study which illustrates how learning a BMRS program amounts to learning *phonological generalizations*.

Chapter 5 discusses syntactic operators over BMRS program. The first part of this chapter defines composition and simultaneous application as operators over BMRS programs. The second part of the chapter shows how a program can be *decomposed* into two individual programs that it is the composition/simultaneous application of. Section 5.3.2 then shows how the learning procedure in Chapter 4 can be extended to learning non-interacting function composition via the syntactic program decomposition procedure.

Chapter 6 explores the themes of *representation and expressivity*. This chapter gives a BMRS characterizations of weakly deterministic functions in terms of the simultaneous application operator defined in Chapter 5. This first part of this chapter presents various cross-linguistic phonological maps which exemplify weakly deterministic (WD) and unbounded circumambient (UC) maps, and proposes a formal characterization of the WD class. The second half of the chapter shows that this characterization succeeds in separating WD and UC maps, and compares these results to to previous attempts at defining WD maps from Heinz and Lai [2013] and Meinhardt et al. [2021].

2

MODEL-THEORETIC PRELIMINARIES

Phonological words are ordered sequences of sounds with internal structure; phonological maps are functions which operate over these structures. The purpose of this chapter is to introduce the model-theoretic foundations in which *representations* of phonological words and maps are formalized.[1] Section 2.1 defines model-theoretic concepts which are used to represent phonological words as string models with predicates that represent features. This section defines signatures, models, subfactors, and logical transductions. Section 2.2 then presents the contemporary formalism of Boolean Monadic Recursive Schemes (BMRS) which are used to represent logical transductions over string models with an abstract programming language. This section presents the syntax of BMRS programs, discusses the evaluation of recursive predicates using fixed point semantics, and presents programs with copy sets. These concepts will ultimately be used to show how BMRS programs can represent phonological maps.

[1]This chapter assumes knowledge of First Order Logic; relevant background can be found in Enderton [1972].

2.1 String Models

Given an alphabet Σ, a string/word is a finite concatenation of symbols in Σ. The set of all such strings, along with the empty string ϵ, is denoted Σ^*. For each $w \in \Sigma^*$, we let w_i denote the i^{th} character in w, and $|w|$ denote the length of w. This section presents definitions of relational structures and string models which are used to represent words in Σ^*, and ultimately functions over Σ^*. These structures are then enhanced with phonological features over which phonological maps operate. The structures that are considered here are very simple word models with a single linear ordering relation on the individual segments of a word. However, these structures can be further enhanced to represent autosegmental tiers and tree structures.

Definition 2.1. A **(relational) signature** is a collection of relation symbols $R_1, \ldots, R_m$, and function symbols $f_1, \ldots, f_n$.

What we refer to here as a 'signature' is commonly referred to in model theory literature as 'vocabulary' (e.g. Libkin, 2004) or 'language' (e.g. Marker, 2002). Signatures can also include a collection of constant symbols; a relational signature is specifically one which does not have constant symbols. The signature only specifies which *symbols* are used to build structures. A structure over a signature $\mathcal{S}$ (called an $\mathcal{S}$-structure) consists of a domain of elements, and an *interpretation* of the relation and function symbols in $\mathcal{S}$ over the domain, as in Definition 2.2.

Definition 2.2. Given a relational signature $\mathcal{S} = \langle \{R_i\}_{i \leq m}, \{f_i\}_{i \leq n} \rangle$, an **$\mathcal{S}$-structure**

$$\mathcal{M} = \langle D, \{R_i^{\mathcal{M}}\}_{i \leq m}, \{f_i^{\mathcal{M}}\}_{i \leq n} \rangle$$

consists of a domain D of elements, a relation $R_i^{\mathcal{M}} \subseteq D^k$ for each k-ary relation symbol R_i, and a function $f^{\mathcal{M}} : D^k \to D$ for every k-ary function symbol f. For a signature $\mathcal{S}$, we denote the set of $\mathcal{S}$-structures as $Struc(\mathcal{S})$.

For an alphabet Σ, we want to construct structures which represent *words* in Σ^*. The signatures for these structures have ordering relations on the individual segments of a word, and predicates (unary relations) for every character in the alphabet, as formalized in Definition 2.3. An alphabet and a signature over that alphabet are often both designated by the same symbol. For this reason, the string models built over an alphabet Σ often called Σ-structures.

Definition 2.3. [String Models] For alphabet Σ and word $w \in \Sigma^*$, a model for w is a Σ-structure

$$\mathcal{M}(w) = \langle D, R_1, \ldots, R_m, \rtimes, \ltimes, P_\sigma \rangle_{\sigma \in \Sigma}$$

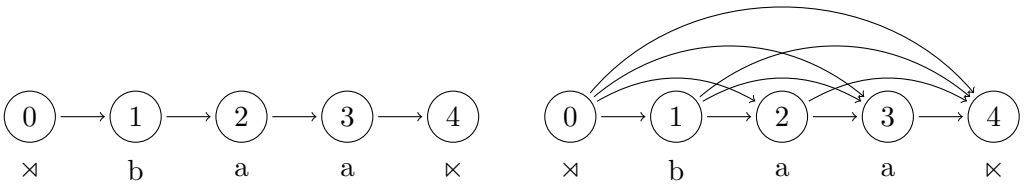
where D is a set of indices $\{0, \ldots, |w| + 1\}$ for each of the characters in the string $\rtimes w \ltimes$, R_i are ordering relations on the elements in D; each $P_\sigma := \{i \in D \,|\, w_i = \sigma\}$ is a unary relation that picks out the elements in D which carry the character $\sigma \in \Sigma$; $\rtimes$ and $\ltimes$ are singleton unary relations such that $\rtimes = \{0\}$ and $\ltimes = \{|w| + 1\}$; for each $i \in \{1, \ldots, |w|\}$, exactly one σ is such that $i \in P_\sigma$.
The **successor model** of w is the structure

$$\mathcal{M}^{\triangleleft}(w) = \langle D, \triangleleft, \rtimes, \ltimes, P_\sigma \rangle_{\sigma \in \Sigma} \quad \text{where} \quad \triangleleft := \{(x, x+1) \in D \times D\}$$

The **precedence model** of w is the structure

$$\mathcal{M}^{<}(w) = \langle D, <, \rtimes, \ltimes, P_\sigma \rangle_{\sigma \in \Sigma} \quad \text{where} \quad < := \{(x, y) \in D \times D \,|\, x < y\}$$

Though the terms 'structure' and 'model' often get used interchangeably, we distinguish string models as a specific case of structure in the following sense: every string model over an alphabet Σ is a Σ-structure, but not vice versa. Successor and precedence models for the string w='baa' over the alphabet $\Sigma = \{a, b\}$ are presented in Figure 2.1. These models have domain $D = \{0, 1, 2, 3, 4\}$ and unary relations $P_a = \{2, 3\}$ and $P_b = \{1\}$. Figure 2.1(a) illustrates the successor model $\mathcal{M}^{\triangleleft}(baa)$, and Figure 2.1(b) illustrates the precedence model $\mathcal{M}^{<}(baa)$. The $\rtimes$ and $\ltimes$ unary relations mark the edges of the string by picking out the first and final indices in D.



(a) Successor model $\mathcal{M}^{\triangleleft}(baa)$; arrows labeled $i \to j$ mean $i \triangleleft j$.

(b) Precedence model $\mathcal{M}^{<}(baa)$; arrows labeled $i \to j$ mean $i < j$.

Figure 2.1: Successor and precedence models of the string 'baa'. Both models have domain $D = \{0, 1, 2, 3, 4\}$ and unary relations $P_a = \{2, 3\}$ and $P_b = \{1\}$. The special symbols $\rtimes = \{0\}$ and $\ltimes = \{4\}$ mark the edges of the string.

These models can be further enriched with other ordering relations in order to represent tiers [Lambert and Rogers, 2020, Chandlee and Jardine, 2019b, Jardine, 2017], syllable structure [Strother-Garcia, 2019, 2018, Strother-Garcia and Heinz, 2017], prosodic structure [Dolatian,

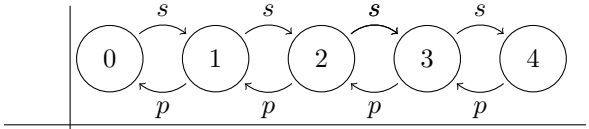
2020], and syntactic trees [Rogers, 2003]. These successor and predecessor models have been used to characterize subregular classes of phonological grammars [Lambert and Rogers, 2020, Rogers and Lambert, 2019, Rogers and Pullum, 2011] and to develop grammatical inference algorithms [Rawski, 2021, Chandlee et al., 2019, Strother-Garcia et al., 2016, Vu et al., 2018].

Recent work on model-theoretic phonology uses a particular kind of successor model in which the ordering relation between elements in the domain is represented with the successor *function* rather than the ordering relation $\lhd$, and the unary relations are represented by Boolean-valued monadic predicates [Chandlee and Jardine, 2021]. Following Chandlee and Lindell (forthcoming), these models are referred to here as 'monadic models'. The monadic model for the string 'baa' is illustrated in Figure 2.2.

Definition 2.4. [**Monadic String Models**] For an alphabet Σ and word $w \in \Sigma^*$, a monadic model for w is a Σ-structure

$$\mathcal{M}(w) = \langle D, \rtimes, \ltimes, P_\sigma, s, p \rangle_{\sigma \in \Sigma}$$

where D is a set of indices $\{0, \ldots, |w| + 1\}$ for each of the characters in $\rtimes w \ltimes$; $P_\sigma : D \to \{\top, \bot\}$ is a monadic predicate such that $P_\sigma(i) = \top$ iff $w_i = \sigma$; $\rtimes(i) = \top$ iff $i = 0$; $\ltimes(i) = \top$ iff $i = |w| + 1$; $s, p : D \to D$ are unary successor and predecessor functions over the elements in D; for each $i \in \{1, \ldots, |w|\}$, there is exactly one σ such that $P_\sigma(i) = \top$.



$\rtimes(x)$	$\top$	$\bot$	$\bot$	$\bot$	$\bot$
$P_a(x)$	$\bot$	$\bot$	$\top$	$\top$	$\bot$
$P_b(x)$	$\bot$	$\top$	$\bot$	$\bot$	$\bot$
$\ltimes(x)$	$\bot$	$\bot$	$\bot$	$\bot$	$\top$
	$\rtimes$	b	a	a	$\ltimes$

Figure 2.2: Monadic model $\mathcal{M}(baa)$; arrows labeled $i \xrightarrow{s} j$ mean $s(i) = j$, and $i \xleftarrow{p} j$ mean $i = p(j)$. Each index has exactly one predicate which evaluates to $\top$ (highlighted).

2.1.1 Phonological Models

The string models discussed so far have the requirement that the predicates $\{\rtimes, \ltimes, P_\sigma\}_{\sigma \in \Sigma}$ partition the domain. In other words, every index of the string carries exactly one character from the set $\Sigma \cup \{\rtimes, \ltimes\}$. This property of string models was highlighted in Figure 2.2. Phonological maps, how-

ever, target particular features, or bundles of features called natural classes [Jakobson et al., 1952, Chomsky and Halle, 1968, Kenstowicz and Kisseberth, 1979, Clements and Hume, 1995]. Consider the collection of consonants /p, v, f, v, t, d, s, z/, the vowel /æ/, and the four binary phonological features [sonorant], [coronal], [continuant], and [voice]. Each of these sounds corresponds with a unique combination of feature specifications, given in Figure 2.3. The sound /d/ for example, has the specification [−son, +cor, −cont, +voi].

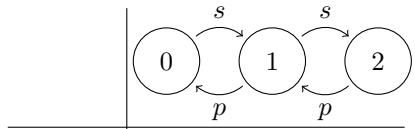
	p	b	f	v	t	d	s	z	æ
[son]	−	−	−	−	−	−	−	−	+
[cor]	−	−	−	−	+	+	+	+	+
[cont]	−	−	+	+	−	−	+	+	+
[voi]	−	+	−	+	−	+	−	+	+

Figure 2.3: Collection of *sounds* and feature specifications for [sonorant], [coronal], [continuant], and [voice], which uniquely identify them.

In order to model phonological words and maps, we use *unconventional* string models [Strother-Garcia et al., 2016], in which more than one predicate can hold at each index. In the case where the predicates represent phonological features, we will refer to these unconventional string models as 'feature models'. For these models, the monadic predicates range over an alphabet of phonological features $\mathbb{F}$, rather than an alphabet of characters/symbols. For simplicity, we use the same notation for each feature $F \in \mathbb{F}$ as the associated monadic predicate. That is, $F(x) = \top$ means the sound at index x of the model is [+F], and $F(x) = \bot$ means the sound at index x of the model is [−F]. These models therefore assume that phonological features are binary. However, Chandlee and Jardine [2021] show that feature models can also be adapted for non-binary feature systems. Further discussion of phonological feature systems from a model-theoretic perspective can also be found in Nelson [2022].

Figure 2.4 presents the feature model for the word /bæd/ over the four features presented in Table 2.3. The sound at index 2 satisfies both the predicates [cor](x) and [voi](x). Since the feature specification [−son, +cor, −cont, +voi] uniquely identifies the sound /d/, the feature model in Figure 2.4 encodes the fact that the sound at index 2 must be a /d/. Thus, each index is associated with a feature matrix that uniquely identifies a sound, rather than a particular character from an alphabet. Moreover, because the requirement that only one predicate hold at each index is dropped for feature models, the predicates INITIAL/FINAL or MIN/MAX can be used in place of the boundary symbols $\rtimes/\ltimes$. The model in Figure 2.4, and the remainder of this dissertation, uses INITIAL and

FINAL predicates to mark the initial and final indices of the word.



	0	1	2
INITIAL(x)	⊤	⊥	⊥
[son](x)	⊥	⊤	⊥
[cor](x)	⊥	⊤	⊤
[cont](x)	⊥	⊤	⊥
[voi](x)	⊤	⊤	⊤
FINAL(x)	⊥	⊥	⊤
	b	æ	d

Figure 2.4: Unconventional string model for /bæd/. The predicates represent (binary) phonological features; $F(x) = \top$ iff the sound at index x is [+F]. Unlike the string model in Figure 2.2, more than one predicate can be true at each index.

In order to present feature models more compactly, we follow Chandlee et al. [2019] in using feature matrices to represent feature models. The feature model for the word /bæd/ in Figure 2.4, for example, is represented with the shorthand notation in (1). This shorthand notation will be used extensively in Chapter 4.

(1) *Shorthand notation for the feature model of /bæd/ in Figure 2.4.*

$$\begin{bmatrix} -\text{son} \\ -\text{cor} \\ -\text{cont} \\ +\text{voi} \end{bmatrix} \begin{bmatrix} +\text{son} \\ +\text{cor} \\ +\text{cont} \\ +\text{voi} \end{bmatrix} \begin{bmatrix} -\text{son} \\ +\text{cor} \\ -\text{cont} \\ +\text{voi} \end{bmatrix}$$

2.1.2 Subfactors

The substructures which are relevant to this dissertation are *subfactors* of successor/monadic unconventional string models. This section therefore does not delve into formal definitions of subfactors in general; discussions and definitions of subfactors can be found in Rogers and Lambert [2019], Chandlee et al. [2019], Strother-Garcia et al. [2016]. Because we only consider successor/monadic models throughout this dissertation, the result which characterizes the relevant notion of 'subfactor' is that subfactors correspond with substring models [Chandlee et al., 2019] (presented in Proposition 2.5). The substrings of $w \in \Sigma^*$ are the strings $v \in \Sigma^*$ for which there exist $u_1, u_2 \in \Sigma^*$ such that $w = u_1 v u_2$. For example, the substrings of 'bac' are $\{\epsilon, b, a, c, ba, ac, bac\}$. Proposition 2.5 says that the *subfactors* of a string model $\mathcal{M}^{\triangleleft}(w)$ are the models $\mathcal{M}^{\triangleleft}(v)$ where v is a substring of w. Proposition 2.5 naturally carries over to monadic models because the sucessor and predecessor functions encode the same ordering relation as $\triangleleft$. In the case of *precedence* models, the subfactor

relation corresponds with subsequences rather than substrings.

Proposition 2.5. [Chandlee et al., 2019] For alphabet Σ and words $v, w \in \Sigma^*$, $\mathcal{M}^{\triangleleft}(v)$ is a **subfactor** of $\mathcal{M}^{\triangleleft}(w)$ (denoted $\mathcal{M}^{\triangleleft}(v) \sqsubseteq \mathcal{M}^{\triangleleft}(w)$) if and only if v is a substring of w.

A subfactor is called a k-factor if the domain has cardinality k. For a model $\mathcal{M}(w)$, the set of subfactors is denoted $Subfact(M(w))$, and the set of k-factors is denoted $Subfact_k(M(w))$. A further result that will be of significance to Chapter 4 is that for any string model $\mathcal{M}$, the structure $(Subfact(\mathcal{M}), \sqsubseteq)$ is a partially-ordered set (Definition 2.6).

Definition 2.6. A structure $(X, \leq)$ forms a **partially-ordered set (poset)** iff $\leq$ is such that

(i) for all $x \in X$, $x \leq x$ [*Reflexivity*]

(ii) for all $x, y \in X$, if $x \leq y$ and $y \leq x$ then $x = y$ [*Anti-symmetry*]

(iii) for all $x, y, z \in X$, if $x \leq y$ and $y \leq z$ then then $x \leq z$ [*Transitivity*]

Consider again the string 'bac' and the set of substrings $\{\epsilon, b, a, c, ba, ac, bac\}$. This set, along with the substring relation, forms a poset. It follows immediately from Proposition 2.5 that $(Struc(\mathcal{M}(bac), \sqsubseteq)$ also forms a poset. The relationship between the substring relation and the subfactor relation for string models is illustrated in Figure 2.5 using the subfactors of $\mathcal{M}^{\triangleleft}(bac)$.

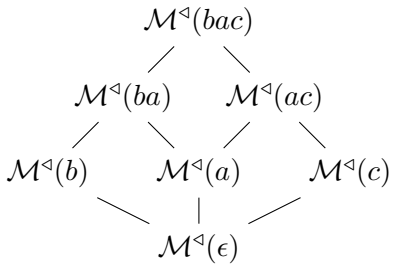


Figure 2.5: Illustration of Proposition 2.5 with the poset $(Subfact(\mathcal{M}^{\triangleleft}(bac)), \sqsubseteq)$.

The subfactors of *unconventional* string models, however, are more complex than simple substrings because any number of predicates can be true at each index. To illustrate the subfactors of an unconventional string model, we consider the weight-stress models from Strother-Garcia et al. [2016], which are structures over the signature $\langle \triangleleft, \texttt{heavy}, \texttt{light}, \texttt{stress} \rangle$. For models over this signature, the individual elements in the domain represent *syllables*. In other words, an n-syllable word will be represented by a model that has a domain of cardinality n. Each syllable can be either be heavy (H), light (L), or unspecified (σ), and each syllable can have the additional property of

being stressed (Ĥ, Ĺ, $\acute{\sigma}$). Thus, there are six possible models of single-syllable words over this signature, illustrated in Figure 2.6.

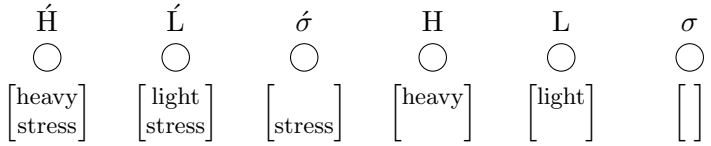


Figure 2.6: Possible models of single-syllable words over the signature $\langle \lhd, \texttt{heavy}, \texttt{light}, \texttt{stress} \rangle$ [Strother-Garcia et al., 2016]. Each syllable can be heavy (H), light (L), or unspecified for weight (σ). In unconventional word models, each syllable can have the additional property of being stressed.

The syllable Ĥ has two properties: heavy and stressed. Subfactors of Ĥ corresponds with one or more of these properties being unspecified. H and $\acute{\sigma}$ are both subfactors of Ĥ; H is the subfactor in which the stress property is unspecified and $\acute{\sigma}$ is the subfactor in which weight is unspecified. Similarly, σ is a subfactor of both H and $\acute{\sigma}$ because it is unspecified for both weight and stress. The six types of structures in Figure 2.6 can therefore be ordered with respect to each other in terms of the subfactor relation on (unconventional) string models, as in Figure 2.7. Similar to Figure 2.5, this structure is a poset.

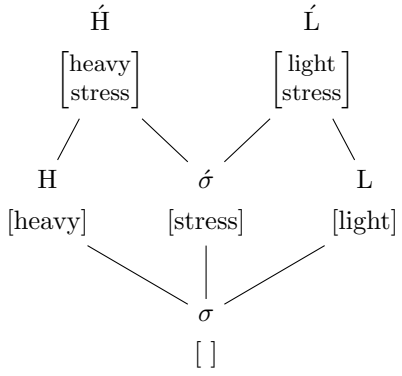


Figure 2.7: Poset corresponding with single-syllable word structures (Figure 2.6), ordered in terms of the subfactor relation on unconventional string models.

The set of subfactors for unconventional string models is significantly larger than the space of subfactors for conventional string models. Consider a word with three syllables where the first is a stressed heavy syllable, followed by two unstressed light syllables (ĤLL). As a conventional string model, the largest subfactors (i.e. substrings) of this string are ĤL and LL, which are also the only 2-factors. In an unconventional model over the signature $\langle \lhd, \texttt{heavy}, \texttt{light}, \texttt{stress} \rangle$, the largest subfactors are {HLL, ĤσL, ĤLσ, $\acute{\sigma}LL$}. Moreover, there are 10 2-factors: {ĤL, Ĥσ, HL,

Hσ, σ́L, σL, LL, Lσ, σ́σ, σσ}. If we add the extra conditions that every word must have at least one syllable, and at least one syllable in every word must be stressed, then the space of possible subfactors of ĤLL is presented in Figure 2.8. Unconventional string models for weight and stress will be revisited in Chapter 6 in a discussion of default-to-same [Hayes, 1995] stress patterns.

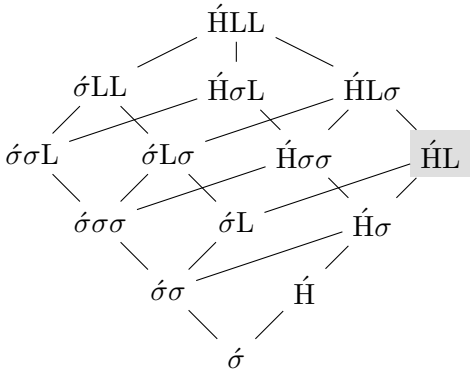


Figure 2.8: Subfactors in an unconventional string model of the three-syllable word ĤLL which satisfy the two requirements: every word must have at least one syllable, and at least one syllable in every word must be stressed. In a conventional string model, the only subfactor that meets these requirements is ĤL (highlighted).

If we consider feature models over the four features $\mathbb{F} = \{[son], [cor], [cont], [voi]\}$ presented in Figure 2.3, the space of subfactors is even larger. However, these large posets of subfactors allow for a formal representation of relevant linguistic generalizations. A model with a single element in the domain and feature specification [-son,+cor,-cont,+voi] represents the sound /d/. An example of a subfactor of this would be a model with no specification for [voice]: [-son,+cor,-cont]. Such a structure no longer represents a particular sound, but instead the set of sounds {/d/, /t/}. Similarly, if we also remove a specification for the feature [continuant] (i.e. [-son,+cor]) we get the set of sounds {/d/, /t/, /s/, /z/}. The linguistic significance of this is that the subfactors represent *natural classes*, and therefore allow us to define phonological maps within the model-theoretic framework in a way that encodes linguistic generalizations. For example, if a phonological map targets voiceless consonants that are preceded by a voiced consonant (as in the case of voicing assimilation), that process targets subfactors of the form [-son,+voi][-son,-voi]. The complete poset of subfactors of /d/ is presented in Figure 2.9. This is discussed further in the context of learning phonological maps in Chapter 4.

2.1.3 Maps

The final piece of the model-theoretic preliminaries necessary for this dissertation is the representation of string *functions*. The structures defined so far in this section represent words. When

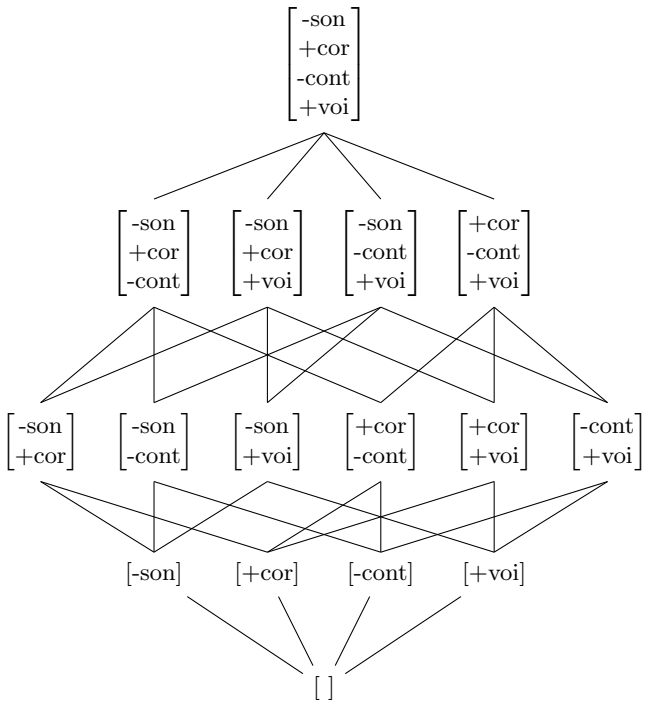


Figure 2.9: Subfactors of the model for the sound /d/, over the set of feature $\{[son], [cor], [cont], [voi]\}$. Each of the subfactors corresponds with a natural class.

these structures are further enriched with phonological features, they represent words in natural language. The focus of this dissertation, however, is on phonological *maps*, which are the relationship between the underlying and surface representations of words in natural language. We represent these maps as relations between their corresponding string models. Consider a string function $f : \Sigma^* \to \Gamma^*$. For every input string $w \in \Sigma^*$ and corresponding Σ-structure $\mathcal{M}(w)$, there is an output string $f(w)$ and corresponding Γ-structure $\mathcal{M}(f(w))$. We capture the relationship between input and output string models with a map that transforms Σ-structures into Γ-structures by means of logical formulas. In particular, the unary relations $\{P_\gamma\}_{\gamma\in\Gamma}$ in the output Γ-structure are associated with logical formulas over the unary relations $\{P_\sigma\}_{\sigma\in\Sigma}$ in the input Σ-structure. These maps are called *logical transductions*, and are a special case of *interpretations* over string models [Courcelle, 1994, Engelfriet and Hoogeboom, 2001, Courcelle and Engelfriet, 2012, Filiot, 2015].

Consider the simple string function f expressed as the rewrite rule $a \to b/b\_\_$, which says: an 'a' in the input string will become a 'b' in the output string if it is immediately preceded by a 'b' in the input. The logical transducer which computes f is given by the function π^f in (2). Since f is a length-preserving function, the input and output models have the same domain, linear ordering,

and boundary predicates $\rtimes$ and $\ltimes$. Moreover, because the input and output alphabets of f are the same, both models have predicates for the alphabet $\{a, b\}$. However, the predicates P_a^f and P_b^f in the output model are expressed as logical formulas over the *signature of the input model*. P_a^f, for example, holds of any index x if and only if P_a holds of that index, and P_b does not hold of the index which immediately precedes x. Within successor models, this is expressed by the existential formula that is highlighted in (2).

(2) *The function $f : a \to b/b\_\_$ as a logical transduction over successor models*

$\pi^f : \langle D, \lhd, \rtimes, \ltimes, P_a, P_b\rangle \to \langle D, \lhd, \rtimes, \ltimes, P_a^f, P_b^f\rangle$, where:

$$P_a^f(x) = P_a(x) \wedge \neg \exists y\big(y \lhd x \wedge P_b(y)\big)$$

$$P_b^f(x) = P_b(x) \vee \big(P_a(x) \wedge \exists y(y \lhd x \wedge P_b(y))\big)$$

An example for the string transformation *baa* $\mapsto$ *bba* is presented in Figure 2.10. The input model $\mathcal{M}^{\lhd}(baa)$ has the predicate $P_a = \{2, 3\}$, as previously presented in Figure 2.1(a). Since $x = 3$ is the only index in the domain which satisfies the FOL formula for $P_a^f(x)$, the output model has predicate $P_a^f = \{3\}$. Similarly, the input model has the predicate $P_b = \{1\}$. Since indices 1 and 2 both satisfy the FOL formula $P_b^f(x)$, the output model has the predicate $P_b^f = \{1, 2\}$. The logical formulas therefore capture the fact that index 2 of the input string model carries an 'a' while index 2 of the output string model carries a 'b'. Thus, the output structure is a model of the string 'bba'. The map π^f therefore transforms the structure $\mathcal{M}^{\lhd}(baa)$ into the structure $\mathcal{M}^{\lhd}(bba)$.

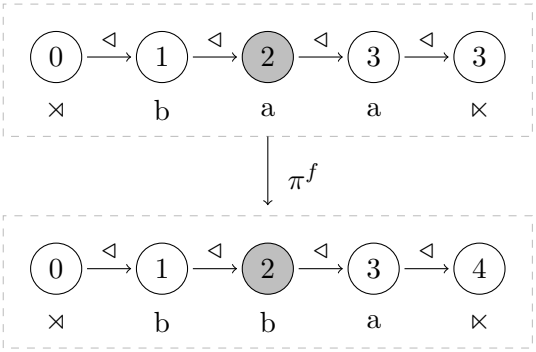


Figure 2.10: The logical transduction in (2), which represents the string function $f : a \to b/b\_\_$, transforms the structure $\mathcal{M}^{\lhd}(baa)$ into the structure $\mathcal{M}^{\lhd}(f(bba))$.

A similar logical transduction can be defined over *monadic* string models. In this case, we use the predecessor and successor functions instead of the linear ordering relation $\lhd$. One significant difference between logical transductions over successor models and monadic models is that the resulting logical formulas over monadic models are *quantifier-free* [Chandlee and Lindell, forth., Chandlee and Jardine, 2019a,b]. For instance, the existential $\exists y(y \lhd x \wedge P_b(x))$ over successor

models states exactly the same property as the expression $P_b(p(x))$ over monadic models. Thus, the logical transduction in (2) computes exactly the same string function as (3). This dissertation uses the language of Boolean Monadic Recursive Schemes (BMRS) to express quantifier-free logical transductions over monadic models. This language is introduced in the next section.

(3) *The function $f : a \to b/b\_\_$ as a quantifier-free logical transduction over monadic models*

$\pi^f : \langle D, \rtimes, \ltimes, P_a, P_b, s, p\rangle \to \langle D, \rtimes, \ltimes, P_a^f, P_b^f, p, s\rangle$ where:

$$P_a^f(x) = P_a(x) \wedge \neg\, P_b(p(x))$$
$$P_b^f(x) = P_b(x) \vee \big(P_a(x) \wedge P_b(px)\big)$$

One final note that is relevant for logical transducers is the evaluation of expressions such as $P_b(p(x))$ in (3) when $x = 0$. This is discussed in Remark 2.7.

Remark 2.7 (**Index Ordering Convention**)**.** The successor and predecessor functions in the monadic string models in Definition 2.4 are partial functions; the initial index does not have a predecessor and the final index does not have a successor. There are other conventions for dealing with the edges of strings so that $p(x)$ and $s(x)$ are never undefined. One alternative would be to define predecessor and successor so that they loops at the edges. In this case, $p(0) = 0$ rather than being undefined. This ordering is adopted by Chandlee and Lindell [forth.]. In order to deal with the edges of strings, we instead adopt the following convention for evaluating a predicate: if x is undefined, we say that $P(x) = \bot$ for every predicate P. Thus $P_b(p(0)) = \bot$ by convention because $p(0)$ is undefined. This convention simplifies how programs are written and evaluated because we do no have to explicitly state extra conditions for the initial and final indices. In order to make the programs in this and subsequent chapters compact and readable, the predicates $\rtimes$/`initial` and $\ltimes$/`final` are reserved for processes that take place at word-initial or word-final positions. However, all the equations in this dissertation can be rewritten to avoid using this convention if necessary. A further advantage for using this convention is that it creates a natural translation between successor and monadic models. For example, $P(p(x))$ over monadic models is logically equivalent to $\exists y(y \lhd x \wedge P(y))$ over successor models with the convention adopted here. If we instead assume that $p(0) = 0$, then these two expressions are no longer logically equivalent.

2.2 Boolean Monadic Recursive Schemes

The formal language of Boolean Monadic Recursive Schemes (BMRS) is a restricted case of recursive program schemes [Moschovakis, 2019], which are constructed using only unary (monadic) Boolean-valued functions. BMRS are used to define logical transductions, which are represented by *programs* in an abstract programming language [Bhaskar et al., 2020, 2023]. Over the unconventional monadic string models discussed in the previous section, these programs are used to model phonological maps [Chandlee and Jardine, 2021, Oakden, 2021, Jardine and Oakden, 2023, McCollum and Jardine, 2022]. This section gives a detailed presentation of the BMRS language by introducing the basic syntax of programs, evaluation of recursive programs, and the use of copy sets.

2.2.1 Programs

The syntax of BMRS expressions is formally described in (4); (a) describes how *index-valued terms* are constructed; (b) describes how *Boolean-valued expressions* are constructed. Further discussion of the syntax and semantics of BMRS and interpretation of programs can be found in Bhaskar et al. [2020, 2023]. Terms are constructed as follows: a variable x which denotes an index value is a term; for any term T, $s(T)$ and $p(T)$ are terms; nothing else is a term. Terms are used to denote elements in the domain of a model over the input signature. Programs are defined using Boolean-valued *expressions*, which are constructed as follows: $\top$ and $\bot$ are expressions; if T is a term and P is a monadic predicate then $P(T)$ is an expression; if f is a recursive unary function then $f(T)$ is an expression; if E_1, E_2, and E_3 are expressions then `if` E_1 `then` E_2 `else` E_3 is an expression; nothing else is an expression. For any term T and monadic predicate P, $P(T)$ is an expression which states whether or not property P holds at index t. These predicates can come from both the input and output signature, thus allowing for recursive expressions. We may also have predicates which are used to mark the edges of the string, such as $\rtimes$ and $\ltimes$. These are left out of the following discussion on syntax of programs because they are ultimately treated in the same way as predicates. The output signature may also have a collection of unary recursive functions which take a term as input and output a Boolean value; these auxiliary functions are discussed in the context of recursion in Section 2.2.2.

(4) *Syntax of BMRS programs with input signature* $\langle P_\sigma, s, p\rangle_{\sigma\in\Sigma}$
and output signature $\langle P_\gamma, f_1, \ldots, f_n, s, p\rangle_{\gamma\in\Gamma}$

a. *Index-valued terms, where* x *is a variable that ranges over* D
$T \ \rightarrow \ x \mid p(T) \mid s(T)$

b. *Boolean-valued expressions*
$E \ \rightarrow \ \top \mid \bot \mid P_\sigma(T)\ (\sigma \in \Sigma) \mid P_\gamma(T)\ (\gamma \in \Gamma) \mid f_i(T) \mid$ if E then E else E

A BMRS *program* with input signature $\langle P_\sigma, s, p\rangle_{\sigma\in\Sigma}$ and output signature $\langle P_\gamma, f_1, \ldots, f_n, s, p\rangle_{\gamma\in\Gamma}$ is the system of unary Boolean-valued equations $\{P_\gamma, f_1, \ldots, f_n\}$. For any string model over the input signature, its corresponding output model is the *solution* to the system of equations (program). The syntax in (4) is for a general case of BMRS programs which model string functions $\Sigma^* \rightarrow \Gamma^*$. The remainder of this chapter and dissertation, however, will focus on string functions where the input and output alphabet are the same.

A crucial component of the BMRS language is the `if...then...else` syntax, which can be used to define propositional logic operators. For any expressions ϕ and ψ, their conjunction, disjunction, and negation are defined in (5). To simplify BMRS expressions, we will often use propositional logic operators for shorthand.

(5) *Logical operations defined using the* `if...then...else` *syntax*
$\phi(x) \wedge \psi(x) \ := \text{if } \phi(x) \text{ then } \psi(x) \text{ else } \bot$
$\phi(x) \vee \psi(x) \ := \text{if } \phi(x) \text{ then } \top \text{ else } \psi(x)$
$\neg\phi(x) \ := \text{if } \phi(x) \text{ then } \bot \text{ else } \top$

We consider first non-recursive programs: those which only reference predicates from the input signature. Two examples of non-recursive programs are presented here to illustrate how BMRS programs represent string functions and phonological maps over feature representations. Example 2.8 revisits the function $f : a \rightarrow b/b\_\_$ from (2) and (3) as a BMRS program. Example 2.9 presents a program that models an obstruent devoicing map over feature models.

Example 2.8 (BMRS Program over String Models)**.** This example gives a BMRS implementation of the string function f expressed with the rewrite rule $a \rightarrow b/b\_\_$, previously discussed in Section 2.1.3. This function is modeled by the BMRS program in (6), where the input signature has predicates $\{P_a, P_b\}$ and the output signature has predicates $\{P_a^f, P_b^f\}$.

(6) *The function* $f : a \rightarrow b/b\_\_$ *as a BMRS program*
$P_a^f(x) \ = \text{if } P_b(p(x)) \text{ then } \bot \text{ else } P_a(x)$
$P_b^f(x) \ = \text{if } P_a(x) \text{ then } P_b(p(x)) \text{ else } P_b(x)$

Consider first the expression for $P_a^f(x)$. For each index x of the input model, whether or not it will output as an 'a' is determined by the `if...then...else` expression. Evaluation of this expression is as follows. Check first whether $P_b(p(x))$ holds (i.e. the 'if' part of the expression). If it does, then $P_a^f(x)$ evaluates to $\bot$ (the 'then' part of the expression). Otherwise, $P_a^f(x)$ evaluates to $P_a(x)$ (the 'else' part of the expression). This `if...then...else` expression therefore gives the following instructions for determining whether x will have character 'a' in the output: If x is preceded by a b in the input model, then x will *not* be an 'a' in the output; otherwise, it will be an 'a' in the output if and only if it is an 'a' in the input. The expression $P_b^f(x)$ similarly states whether the output string will have a 'b' character at index x; if x is an 'a' in the input, then it will output as 'b' if it is preceded by a 'b'; otherwise, it will output as 'b' if it is a 'b' in the input. Note that the predicates $P_a^f(x)$ and $P_b^f(x)$ in (6) are logically equivalent to the predicates that were defined in (3) using conjunction and disjunction.

The logical transduction in (6) is presented over the string model for 'baa' in Figure 2.11. Because $P_b(p(2)) = P_b(1) = \top$, the 'if' part of the expression holds for $x = 2$, and therefore $P_a^f(2)$ evaluates to $\bot$. This is highlighted in 3^{rd} row of Figure 2.11. Similarly, because $P_a(2) = \top$, $P_b^f(2)$ evaluates to $P_b(p(2)) = P_b(1) = \top$. This is highlighted in the 4th row of Figure 2.11. The output string 'bba' is the *solution* to the equations in (6); it is the unique string which satisfies the collection of truth values in Figure 2.11. For example, $P_a^f(2) = \bot$ and $P_b^f(2) = \top$ means that index 2 must have a 'b' character in the output. Thus, the program in (6) yields the *baa* $\mapsto$ *bba* string transformation.

input	⋊	b	a	a	⋉
	0	1	2	3	4
$P_a(x)$	$\bot$	$\bot$	$\top$	$\top$	$\bot$
$P_b(x)$	$\bot$	$\top$	$\bot$	$\bot$	$\bot$
$P_a^f(x)$	$\bot$	$\bot$	$\bot$	$\top$	$\bot$
$P_b^f(x)$	$\bot$	$\top$	$\top$	$\bot$	$\bot$
output	⋊	b	b	a	⋉

Figure 2.11: System of equations in (6) for the string function $f : a \to b/b\_\_$ applied to the input string model for 'baa'. In the input model, $P_a(2) = \top$ and $P_b(2) = \bot$. In the output model, the truth values flip (highlighted). Similar to Figure 2.10, the character at index 2 changes from an 'a' in the input string to a 'b' in the output string.

Example 2.9 (BMRS Program over Feature Models)**.** This example gives a BMRS implementation of word-final obstruent devoicing. This function is modeled by the program in (7). Since final-obstruent devoicing involves a change at the end of the word, we need a way to encode the word

boundary on the right end of the string. As discussed in Section 2.1.1, we use the predicates `initial` and `final` in feature models rather than $\rtimes$ and $\ltimes$.

(7) *Final obstruent devoicing* $[-son, +voi] \rightarrow [-voi]/\_\_\#$ *as a program over feature models*

$$
\begin{aligned}
[son]'(x) &= [son](x) \\
[cor]'(x) &= [cor](x) \\
[cont]'(x) &= [cont](x) \\
[voi]'(x) &= \text{if } \big(\neg[son](x) \wedge \texttt{final}(x)\big) \text{ then } \bot \text{ else } [voi](x)
\end{aligned}
$$

The expression for $[voi]'(x)$ states when x will be [+voice] in the output. The expression $(\neg[son](x) \wedge \texttt{final}(x))$ (the 'if' part of $[voi]'(x)$) expresses the property of being a word-final obstruent. If this holds of x then $[voi]'(x)$ evaluates to $\bot$, meaning x will be voiceless in the output. Otherwise, x evaluates to $[voi](x)$, meaning it is voiced in the output iff it is voiced in the input. Since none of the sounds undergo change with respect to the remaining three features, the output predicates simply evaluate to the same value as the input predicates for these two features.

Figure 2.12 presents the program in (7) over the underlying form /bæd/, which was presented in Figure 2.4. For brevity, only the relevant predicates are listed in the table. Because $\texttt{final}(2) = \top$ and [son](2)= $\bot$, the 'if' part of the definition of $[voi]'(2)$ holds. Thus, $[voi]'(2)$ evaluates to $\bot$ and the sound at index 2 will be voiceless in the output. This is highlighted in the final row of the table in Figure 2.12. Since the sound at index 2 does not undergo any change with respect to any other features, it must output as the voiceless [t]. Since $\texttt{final}(x)$ does not hold of any other indices, they evaluate to $[voi](x)$ and therefore remain faithful to their input specification. The program in (7) therefore yields the /bæd/$\mapsto$[bæt] transformation.

input	b	æ	d
	0	1	2
[son](x)	$\bot$	$\top$	$\bot$
[cor](x)	$\bot$	$\top$	$\top$
[cont](x)	$\bot$	$\top$	$\bot$
[voi](x)	$\top$	$\top$	$\top$
`final`(x)	$\bot$	$\bot$	$\top$
[voi]$'(x)$	$\top$	$\top$	$\bot$
output	b	æ	t

Figure 2.12: System of equations in (7), applied to the underlying form /bæd/. [voi](2) = $\top$ in the input model; [voi]$'(2) = \bot$ in the output model. The sound at index 2 thus changes from having feature specification [−son,+cor,−cont, +voi] to [−son,+cor,−cont, −voi], yielding the surface form [bæt].

2.2.2 Recursion

Examples 2.8 and 2.9 both discuss functions that require *local information in the input.* For example, the function $f : a \to b/b\_\_$ in Example 2.8 transforms an 'a' into a 'b' if it is *immediately preceded* by a 'b'. Consider, on the other hand, the long-distance version of that function: $f^* : a \to b/bx^*\_\_$, where $x^* \in \Sigma^*$. This function transforms an 'a' into a 'b' if it is preceded by a 'b' *anywhere on the left.* The logical transducer which represents this function is given in (8). The difference between this and the transducer defined in (2) is that it uses the $<$ relation rather than the $\lhd$ relation.

(8) *The function $f^* : a \to b/bx^*\_\_$ as a logical transduction over precedence models*
$\pi^{f^*} : \langle D, <, \rtimes, \ltimes, P_a, P_b\rangle \to \langle D, <, \rtimes, \ltimes, P_a^{f^*}, P_b^{f^*}\rangle$ where:
$P_a^{f^*}(x) \;= P_a(x) \wedge \neg\exists y\big(\, y < x \,\wedge P_b(y)\big)$
$P_b^{f^*}(x) \;= P_b(x) \vee \big(P_a(x) \wedge \exists y(\, y < x \,\wedge P_b(y))\big)$

Recall that the FOL formula $\exists y(y \lhd x \wedge P_b(y))$ over successor models was equivalent to $P_b(p(x))$ over monadic models. The formula $\exists y(y < x \wedge P_b(y))$, on the other hand, does not have an equivalent expression over monadic models. In order to express f^* as a quantifier-free logical transduction over monadic models, we need *recursive* definitions for the output predicates in a program. This section shows how recursive programs are evaluated as the least fixed point of a monotonic sequence of partial functions [Bhaskar et al., 2020, Moschovakis, 2019]. Example 2.10 first presents an iterated function which uses local information *in the output* and therefore requires programs that are defined in terms of both input and output predicates. This examples demonstrates in detail how a monotone sequence of functions is constructed and evaluated. The discussion of recursion in Example 2.10 is then extended to the function f^* to show how recursion is used to express the FOL formula $\exists y\big(y < x \wedge P_b(y)\big)$ in (8).

Example 2.10 (Iterated Functions)**.** The function $f_{it} : a \to b/b\_\_$ (*iterate*) is the iterated form of the function f from Example 2.8. The difference between these two functions is that f applies to the input once, while f_{it} reapplies until no further changes can be made. Figure 2.13 shows this iterative process over the input string 'baa'. A single application of f yields the output 'bba'. The iterative form of f reapplies the function to this new output to give 'bbb'.

The BMRS system of equations that models the function f_{it} is presented in (9). Notably, it looks exactly like the system of equations for the function f in (6), with the exception that this

input	b	a	a
f_{it}	b	b	a
f_{it}	b	b	b
output	b	b	b

Figure 2.13: Iterative function f_{it} applied to the string 'baa'. The function reapplies until no further changes can be made.

program is defined in terms of the *output* predicate $P_b^{f_{it}}(p(x))$, whereas (6) was defined in terms the input predicate $P_b(p(x))$. This change is highlighted in (9).

(9) *The function* $f_{it} : a \to b/b\_(iterate)$ *as a BMRS program*

$$P_a^{f_{it}}(x) = \text{if } P_b^{f_{it}}(p(x)) \text{ then } \bot \text{ else } P_a(x)$$
$$P_b^{f_{it}}(x) = \text{if } P_a(x) \text{ then } P_b^{f_{it}}(p(x)) \text{ else } P_b(x)$$

Recursive equations like $P_b^{f_{it}}(x)$ are evaluated over a model as the fixed point of a monotone sequence of partial functions. The sequence starts with the empty function f_0, which does not assign a truth value for any of the indices in the domain. The next partial function in the sequence f_1 gives a Boolean value to some subset of the indices. Each subsequent function f_{n+1} in the sequence *extends* f_n by maintaining all the same truth value assignments as f_n and assigning truth values to some indices that f_n did not assign truth values to. Although the sequence of functions $\{f_n\}_{n\geq 0}$ is infinite, there is a smallest value N such that f_N assigns a truth value to all indices in the domain and $f_n = f_N$ for all $n > N$ [Moschovakis, 2019]; this function is the *least fixed point* of the sequence. The monotone sequence of function that we use to evaluate $P_b^{f_{it}}(x)$ is given in (10). The initial function f_0 does not evaluate to $\top$ or $\bot$ for any x. Each subsequent function f_{n+1} is constructed by replacing all occurrences of $P_b^{f_{it}}$ with f_n, as highlighted in (10).

(10) *Monotone sequence of functions used to evaluate* $P_b^{f_{it}}$ *in (9)*

$$f_0 = \emptyset$$
$$f_{n+1}(x) = \text{if } P_a(x) \text{ then } f_n(p(x)) \text{ else } P_b(x)$$

In the case of a *program* consisting of several equations, we get a collection of monotone sequences of functions. In the case of f_{it}, both $P_a^{f_{it}}$ and $P_b^{f_{it}}$ are evaluated as a sequence of functions. However, for simplicity, we focus on $P_b^{f_{it}}$ to illustrate the least fixed point semantics. Figure 2.14 shows how the monotone sequence of functions evaluate $P_b^{f_{it}}$ over the input string 'baa'. The function f_0 does not assign any truth values; the ! symbol is used to denote an undefined truth value. $f_1(x)$ is evaluated as follows: check whether $P_a(x)$ holds; if it does, evaluate $f_0(p(x))$; otherwise,

evaluate $P_b(x)$. Because $P_a(0) = \bot$, $f_1(0)$ evaluates to $P_b(0)$, which is $\top$ (highlighted in 2nd row of Figure 2.14). By the same reasoning $f_n(0) = \top$ for all $n \geq 1$. f_1 is undefined for the remaining two indices because they evaluate to $f_0(p(x))$ which is undefined for the remaining indices x. The function f_2 then *extends* f_1 by assigning a truth value to index 1; because $P_a(1) = \top$, $f_2(1)$ is evaluated as $f_1(0) = \top$. This is highlighted in the third row of Figure 2.14. In a similar manner, f_3 extends f_2 by maintaining all the same truth value assignments as f_2 and adding a truth assignment to index 3. f_3 is the *least fixed point* of the sequence; it is the first function in the infinite sequence $\{f_n\}_{n\geq 0}$ such that all subsequent functions will have the same truth value assignments for all indices. $P_b^{f_{it}}$ is evaluated as the the least fixed point of the sequence in (10) and Figure 2.14. In other words, $P_b^{f_{it}}(x) = f_3(x)$ for every x. Figure 2.15 shows how the program in (9) yields the output 'bbb' for the input 'baa'.

	b	a	a
	0	1	2
$f_0(x)$	!	!	!
$f_1(x)$	$\top$	!	!
$f_2(x)$	$\top$	$\top$	!
$f_3(x)$	$\top$	$\top$	$\top$

Figure 2.14: Monotonic sequence of partial functions in (10), applied to the input string model 'baa'. f_0 is undefined at all indices. f_{n+1} extends f_n by assigning truth value to index n. f_3 is the least fixed point of the infinite sequence of functions $\{f_n\}_{n\geq 0}$.

	b	a	a
$P_a(x)$	$\bot$	$\top$	$\top$
$P_b(x)$	$\top$	$\bot$	$\bot$
$P_a^{f_{it}}(x)$	$\bot$	$\bot$	$\bot$
$P_b^{f_{it}}(x)$	$\top$	$\top$	$\top$
	b	b	b

Figure 2.15: System of equations in (9) for the iterated string function f_{it}, applied to the input string model for 'baa'. The recursively-defined output predicate $P_b{}^{f_{it}}$ is evaluated as the least fixed point of the sequence of functions in (10); $P_b{}^{f_{it}}(x) = f_3(x)$ for all x.

The iterated function f_{it} in Example 2.10 depends on local information in both the input and the *output* and therefore, the output predicate $P_b^{f_{it}}$ is defined as a BMRS formula over both P_b and $P_b^{f_{it}}$. The long-distance version of this function $f^* : a \to b/bx^*\_\_$, on the other hand, depends on information found an *unbounded* distance away in the input (i.e. whether there is a 'b' somewhere to the left of an 'a'). Although words are always finite, we say the information is an 'unbounded' distance away because there is no fixed k-window in which the relevant information can be found. In the logical transducer in (8), the predicate $P_b^{f^*}$ is defined using the FOL formula

$\exists y(y < x \wedge P_b(y))$ which says: there is a 'b' somewhere to the left of x. In order to represent f^* as a BMRS transduction, recursion is used to express the existential expression, as in (11).

(11) *Non-local information in strings*
`b-left`$(x) = \top$ iff there is a 'b' somewhere to the left of index x

a. *As an FOL formula over precedence models*
`b-left`$(x) = \exists y(y < x \wedge P_b(y))$

b. *As a BMRS formula over monadic models*

$$\texttt{b-left}(x) = \text{if } \texttt{initial}(x) \text{ then } \bot \text{ else if } P_b(px) \text{ then } \top \text{ else } \texttt{b-left}(px)$$

Note that the evaluation of partial functions amounts to a three-valued logic, because we have to consider the truth value of an `if...then...else` expressions where one of the expressions has an undefined truth value. The table presents in Figure 2.16 presents the three possible scenarios in which an `if...then...else` expressions has an undefined truth value.

P	Q	R	if P then Q else R
!			!
$\top$	!		!
$\bot$		!	!

Figure 2.16: Three possible scenarios in which an `if...then...else` expression can have an undefined truth value (denoted !).

The BMRS program which models the function f^* is presented in (12). Whereas previous programs only had output predicates for elements of the alphabet, this program also has the recursive function `b-left` in the output signature. Notably, this program looks exactly like the programs for f in (6) and f_{it} in (9), with the exception that occurrences of P_b in (6) and occurrences of $P_b^{f_{it}}$ in (9) are replaced by the auxiliary function `b-left`. This distinction between the three functions is summarized in Figure 2.17.

(12) *The function $f^* : a \to b/bx^*\_\_$ as a BMRS program* (where $x^* \in \Sigma^*$)

$$P_a^{f*}(x) = \text{if } \texttt{b-left}(x) \text{ then } \bot \text{ else } P_a(x)$$

$$P_b^{f*}(x) = \text{if } P_a(x) \text{ then } \texttt{b-left}(x) \text{ else } P_b(x)$$

The three functions in Figure 2.17 are significant for phonology because they are representative of three nested complexity classes of phonological maps that are most commonly found in natural language. These functions are discussed in Chapter 3 with cross-linguistic examples of phonological maps that exemplify each of these classes.

function	output predicate for 'b'
$f : a \to b/b\_\_$	$P_b^f(x) =$ if $P_a(x)$ then $P_b(p(x))$ else $P_b(x)$
$f_{it} : a \to b/b\_\_(iterate)$	$P_b^{f_{it}}(x) =$ if $P_a(x)$ then $P_b^{f_{it}}(p(x))$ else $P_b(x)$
$f^* : a \to b/bx^*\_\_$	$P_b^{f*}(x) =$ if $P_a(x)$ then `b-left`(x) else $P_b(x)$

Figure 2.17: Highlighting the distinction between the function f, f_{it}, and f^*. f depends on local information in the input; the corresponding output predicate is defined over predicates in the input signature. f_{it} depends on local information in both the input and output; the corresponding output predicate is defined over predicates in both the input and output signature. f^* depends on non-local information; the corresponding output predicate is defined using a recursively-defined auxiliary formula.

2.2.3 Copy Sets

The examples presented so far are length-preserving maps. This means that there is a one-to-one correspondence between input and output indices. With respect to logical transducers in general, it means the domain and ordering relation between the input and output models are the same. However, phonological maps are not always length-preserving. The epenthesis map $\emptyset \to b/a\_\_a$, for example, inserts a 'b' between two 'a' characters in the input, and yields an output string that is longer than the input string. The BMRS formalism introduced so far in this section cannot model this function because it does not have a one-to-one correspondence between segments in the input and output strings. However, the general form of BMRS programs which includes a 'copy set' can model functions like this. Copy sets are introduced in this subsection, and the BMRS program which models epenthesis is discussed in Example 2.12. Further discussion of epenthesis maps are logical transductions with copy sets can be found in Chandlee and Lindell [forth.].

The use of copy sets for defining string transductions in BMRS is adapted from Courcelle [1994] and Engelfriet and Hoogeboom [1999, 2001]. Each program has some fixed number k copies of every output predicate. More formally, a 'copy set' is a set $C = \{1, \ldots, k\}$ such that for an input model with domain D, the output model has domain $D \times C$. This means that output predicates are defined over a matrix of indices, presented in Figure 2.18. Each output predicate is defined over a *pair* of numbers (c, x), where c represents the copy set. Because c is not a variable, these predicates are still *monadic*. A BMRS program has a collection of output predicates, parameterized with a copy index.

0	1	2	...	n
$(0,0)$	$(0,1)$	$(0,2)$	...	$(0,n)$
$(1,0)$	$(1,1)$	$(1,2)$	...	$(1,n)$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$(k,0)$	$(k,1)$	$(k,2)$	...	(k,n)

Figure 2.18: Input and output indices. For each pair (c,x), c represents the *copy* set and x corresponds with an index of the input string.

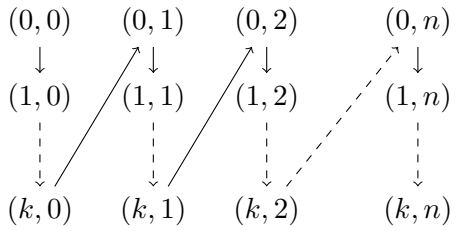


Figure 2.19: Linear ordering on indices of the output model, where arrows labeled $(c,x) \to (c',x')$ mean $s(c,x) = (c',x')$.

The linear ordering on indices presented in Figure 2.19 is a special case of *order-preserving* BMRS programs (Definition 2.11). Order-preserving means that for any index x, all copies of x precede all copies of $x+1$; this is captured by condition (i). Moreover, at any index, copy c precedes copy $c+1$ of the same index; this is captured by condition (ii). While it is possible to define programs which are not order-preserving, doing so increases the expressive power of BMRS. We are particularly interested in the more restrictive case of order-preserving programs because they model a particular class of string functions which are of relevance to phonology. This is discussed further in Section 3.2.

Definition 2.11. Given a model $\mathcal{M}$ with domain D, a BMRS program with copy set C is **order-preserving** iff for every $x, x' \in D$ and copies $c, c' \in C$ the following two properties hold

(i) $(c,x) \leq (c,x')$ iff $x \leq x'$

(ii) $(c,x) \leq (c',x)$ iff $c \leq c'$

Example 2.12 (Epenthesis Function with Copy Sets)**.** The BMRS program which models the epenthesis map $\emptyset \to b / a\_\_a$ is given in (13). The first copy (i.e. the predicates indexed with 0) copy over the input string and do not make any changes. The second copy (index with 1) is used for the epenthesized character. $P'_a(1,x) = \bot$ indicates that an 'a' will never be found in the second copy; this is because an 'a' never gets epenthesized. $P'_b(1,x)$ gives the conditions for when a 'b' character appears in the second copy: if x carries an 'a' and is immediately followed by another 'a'.

(13) *Epenthesis map $\emptyset \to b / a\_\_a$ as a BMRS program with a copy set*

$$
\begin{aligned}
P'_a(0,x) &= P_a(x) \\
P'_b(0,x) &= P_b(x) \\
P'_a(1,x) &= \bot \\
P'_b(1,x) &= P_a(x) \wedge P_a(s(x))
\end{aligned}
$$

Figure 2.20 shows how the program in (13) yields an epenthesized 'b' for the input string 'aa' but not for 'ab'. In both cases, copy set 0 simply copies the entire string over. In Figure 2.20(a), $P_a(0)$ and $P_a(s(0))$ both hold, so $P_b(1,0)$ evaluates to $\top$. Thus, index 0 of copy 1 carries a 'b', which is highlighted in 2.20(a). Similar to Bhaskar et al. [2020], we assume that for any index which has no output predicates evaluate to $\top$, the output at that index is the empty string. In Figure 2.20(a), both $P_a(1,1)$ and $P_b(1,1)$ are $\bot$ and therefore the output at $(1,1)$ is the empty string. With the ordering in Figure 2.19, the resulting output over the input string 'aa' is 'aba'. For the input string 'ab' in Figure 2.20(b), the row for copy set 1 is empty since all the predicates evaluate to $\bot$, and the output is therefore 'ab'.

	a	a
	0	1
$P_a(x)$	$\top$	$\top$
$P_b(x)$	$\bot$	$\bot$
$P'_a(0,x)$	$\top$	$\top$
$P'_b(0,x)$	$\bot$	$\bot$
$P'_a(1,x)$	$\bot$	$\bot$
$P'_b(1,x)$	$\top$	$\bot$
0	a	a
1	b	

(a) Over the input string 'aa', the epenthesized 'b' appears in the second copy of index 0.

	a	b
	0	1
$P_a(x)$	$\top$	$\bot$
$P_b(x)$	$\bot$	$\top$
$P'_a(0,x)$	$\top$	$\bot$
$P'_b(0,x)$	$\bot$	$\top$
$P'_a(1,x)$	$\bot$	$\bot$
$P'_b(1,x)$	$\bot$	$\bot$
0	a	b
1		

(b) Over the input string 'ab', the second copy does not have any output values, so nothing gets epenthesized.

Figure 2.20: System of equations in (13) and resulting outputs for input strings 'aa' and 'ab'. The resulting output strings are determined by the linear ordering presented in Figure 2.19.

The phonological patterns discussed throughout this dissertation are all length-preserving and therefore modeling them with BMRS programs does not require a copy set. However, copy sets must be considered for any formal statements about the expressivity of BMRS. That is, for any claims about what can and *cannot* be modeled with a BMRS program, we must include a discussion of copy sets. Copy sets are crucial for Bhaskar et al. [2020]'s characterization of the subsequential

functions in terms of BMRS programs that are restricted to using only successor or predecessor, and for the characterization of weakly deterministic functions in Chapter 6.

3

COMPUTATION AND LOGIC IN PHONOLOGY

Phonological grammars are regular. This formal language theoretic result means that phonological grammars can be modeled with finite state automata, and consequently provides a formal description of what kinds of logically-possible processes *cannot* be a phonological process in *any* natural language. Phonological grammars are further refined into nested complexity classes called the Subregular Hierarchy. This chapter presents the Subregular Hierarchy of phonological maps, and gives a logical perspective on these maps through the BMRS formalism that was discussed in the previous chapter.[1] Section 3.1 discusses the subregular hierarchy through examples of cross-linguistic maps which exemplify the different regions of the hierarchy. Section 3.2.1 shows how finite state transducers can be translated to *logical* transducers using the language of BMRS, and Section 3.2.2 discusses logical characterizations of the subregular classes of functions in terms of BMRS programs.

[1]This chapter assumes knowledge of finite state transducers (FSTs). Background readings on FSTs can be found in [Mihov and Schulz, 2019, Mohri, 1997]. Background reading on applications of finite state automata to phonology and morphology can be found in Chandlee and Jardine [2022], Hulden [2009], Beesley and Karttunen [2003].

3.1 Phonology and Computation

The claim that phonological grammars are regular embodies two facts: (1) phonological transformations can be expressed by regular *relations* (i.e. those that are computed by finite state *transducers*), and (2) phonotactics can be expressed with regular *stringsets* (i.e. those that are computed by finite state *acceptors*). The statement in (1) is due to the result that rewrite rules in Chomsky and Halle [1968]'s *Sound Patterns of English* (SPE) formalism describe regular relations [Kaplan and Kay, 1994, Johnson, 1972]. Since all phonological maps can be expressed within this formalism, it follows that phonological maps are describable by regular relations. This statement then extends to phonological *grammars*, where the relationship between underlying and surface forms is the composition of a set of ordered rewrite rules. Since the composition of any two regular relations is also a regular relation, this means that phonological *grammars* within the SPE formalism are regular. The statement in (2) is due to a result that the image of a regular relation is a regular stringset [Scott and Rabin, 1959]. If phonological maps are regular relations, then this result means that grammatical surface forms (i.e. phonotactics) are regular sets. Thus, all of natural language phonology falls within the domain of finite state automata. This result provides a computational contrast between the phonological and syntactic components of grammar because natural language syntax is outside of the regular boundary [Chomsky, 1956, 1959, Shieber, 1985]. Particularly, long-distance dependencies in syntax cannot be computed by finite state machines, while long-distance dependencies in phonology *can*. Heinz and Idsardi [2013] argue that the computational difference between phonology and syntax indicates different cognitive learning mechanisms. The result that phonology is regular therefore provides insight into the cognitive complexity of the phonological component of grammar [Heinz and Idsardi, 2013, De Santo and Rawski, 2022, Heinz and Idsardi, 2011, Rogers et al., 2013].

The Subregular Hierarchy [Rogers and Pullum, 2011, Rogers et al., 2013, Chandlee and Heinz, 2018, Heinz, 2018] further refines the regular class into nested regions which describe the expressivity of cross-linguistic phonological patterns. Because the term 'regular' is used for both stringsets and string relations, each of these are refined into a separate subregular hierarchy. The hierarchies discussed by Rogers and Pullum [2011] and Rogers et al. [2013] are for stringsets. These hierarchies are beyond the scope of this dissertation, and further discussion of their relevance to phonology can

be found in Heinz [2018]. Since the topic of this dissertation is phonological *maps*, this chapter only discusses the subregular hierarchy of string *functions*, presented in Figure 3.1. This hierarchy characterizes the expressivity of various phonological patterns in terms of dependencies between sounds in the underlying and surface representations. The strictly local classes ISL and OSL characterize the maps in which surface forms are dependent on *local* information. These classes are contained within the subsequential region, meaning all strictly local functions are left or right subsequential, but not vice versa [Chandlee, 2014]. The left (right) subsequential functions characterize maps in which surface forms are dependent only on information found to the left (right) in the underlying form. The computational significance of this class of functions is that they are computable by deterministic transducers [Mohri, 1997]. The region covered by left and right subsequential functions in Figure 3.1 makes up the *deterministic boundary*.

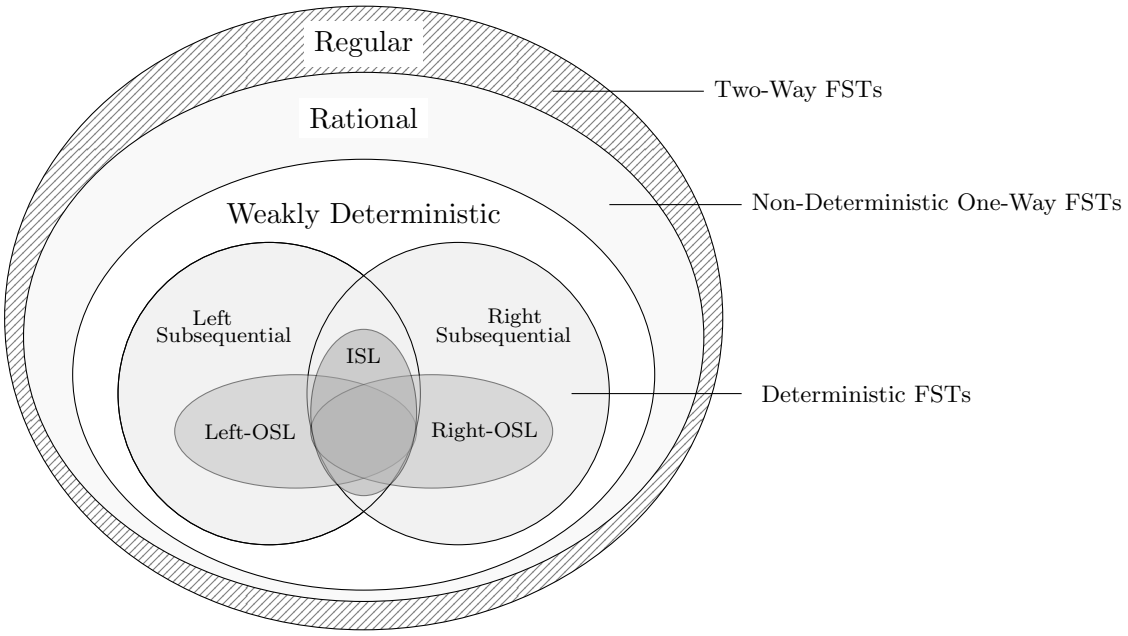


Figure 3.1: Subregular hierarchy of string relations/functions, adapted from Chandlee [2014, pg.57], Jardine [2016a, pg.17], and Heinz [2018, pg.166].

In our discussion of phonological maps, we restrict our focus to the rational class of functions, rather than regular. These are the functions which are computable by *one-way* finite state transducers. The regular functions, on the other hand, are those computable by *two-way* finite state transducers, in which the transducer can move through the input string both left-to-right and right-to-left. Phonological maps fall within the more restrictive rational class. Moreover, what

phonologists often refer to as a 'regular' function is formally a rational function. Remark 3.1 clears up potential confusion over terminology used across various literature cited here. Further discussion of the distinction between rational and regular transductions can be found in Filiot and Reynier [2016], Filiot et al. [2019].

Remark 3.1 (**Variations in Terminology**)**.** The term 'regular' often gets used interchangeably with the term 'rational' for string relations. The functions which are referred to as 'regular' in the computational linguistics literature are often referred to as 'rational' elsewhere. Kaplan and Kay [1994] state: "The relations we refer to as 'regular', to emphasize the connection to formal language theory, are often known as 'rational relations' in the algebraic literature". The reason for the terms being conflated is due the use of 'regular' for *languages* that are recognized by a finite state acceptor. Furthermore, the functions which are referred to as 'subsequential' in computational phonology literature are often referred to as 'sequential' in computer science literature. This dissertation will stick to the convention of using the term 'subsequential' when talking about phonological maps computable by deterministic FSTs, and use 'rational' for the phonological maps computable by non-deterministic FSTs.

Sections 3.1.1 and 3.1.2 discuss the different regions of the subregular hierarchy in Figure 3.1, along with examples of phonological processes from various languages which exemplify each of the classes. The linguistic examples presented throughout this chapter use a consistent notation of **boldface to indicate a sound which has undergone change**, and underlining to indicate the sound(s) which triggered that change.

3.1.1 The Subsequential Hypothesis

Although all phonological grammars are regular, most cases of vowel harmony [Gainor et al., 2012], metathesis [Chandlee et al., 2012], consonant harmony [Luo, 2017], and dissimilation [Payne, 2017] are subsequential. These observations led to the Subsequential Hypothesis [Heinz, 2018, Heinz and Lai, 2013], which proposes a tighter bound on phonological complexity by asserting that phonological maps in natural language are left or right subsequential. The significance of this class of functions within the subregular hierarchy is that it encompasses all the functions that can be computed by a *deterministic* finite state transducer (FST) [Mohri, 1997]. The Subsequential Hypothesis

therefore proposes a *deterministic boundary* on the expressivity of natural language phonological processes. The subsequential boundary is significant for phonology because it has advanced our understanding of the computational nature of phonological processes [Chandlee and Heinz, 2018, Chandlee, 2014, Heinz and Lai, 2013, Luo, 2017, Payne, 2017, Chandlee and Heinz, 2012, Gainor et al., 2012], their implications for cognition [Heinz and Idsardi, 2013, 2011, Rogers et al., 2013] and natural language learning [Lambert et al., 2021, Finley, 2008, 2012, Finley and Badecker, 2008, Lai, 2015], and the development of computational learning algorithms for phonological transformations [Chandlee, 2014, Jardine et al., 2014, Heinz, 2011, 2009, Gildea and Jurafsky, 1995, Oncina et al., 1993].

The subsequential boundary contains several regions: input strictly local, left and right output strictly local, and left and right subsequential. Postnasal voicing in Zoque [Wonderly, 1951] is an example of an ISL function. The data points in (14) show that voiceless stops become voiced when they are preceded by a nasal sound. The voiceless stop /p/ in the word /pama/ 'clothing', for example, is pronounced as a voiced [b] in [mbama] 'my clothing'. In general, the input-output dependency that characterizes ISL functions is illustrated in Figure 3.2(a), where the surface form of each input element depends on information within a neighborhood of fixed size *in the input*. In the case of postnasal voicing, for every x in the input string, its corresponding output depends only on x and the sound immediately preceding x.

(14) **[Input Strictly Local (ISL)]** Postnasal Voicing in Zoque [Wonderly, 1951]

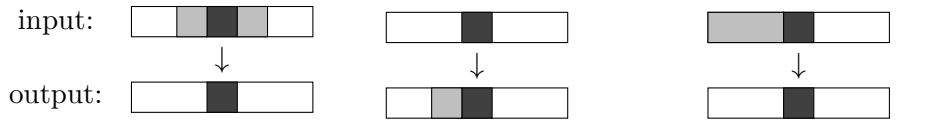
a.		b.	
	pama 'clothing'		m-**b**ama 'my clothing'
	tatah 'father'		n-**d**atah 'my father'
	kama '(corn)field'		ŋ-**g**ama 'my (corn)field'

In contrast, the output strictly local (OSL) functions are those in which the surface form of each element in the input string depends on local information in the *output*. These are split into Left-OSL and Right-OSL depending on whether the necessary information is within a local window to the left or to the right. Nasal harmony in Warao [Osborn, 1996, Piggott and van der Hulst, 1997, Peng, 2000] is an example of a left-OSL process. The selected data points in (15) present nasal harmony, and the blocking of the process by voiceless obstruents. The first pair of words in (15a) share the dative suffix /-ya/. When the root word does not contain any nasal sounds the suffix is not nasalized, as in [esoha-ya] 'he pours'. When the root does contain a nasal, all the

subsequent sounds in the word become nasalized and the dative surfaces as [-ỹã], as in [nãõ-ỹã] 'he saw'. The contrast between [nãõ-ỹã] and [nãõ-te] shows how the voiceless obstruent /t/ blocks harmony from spreading any further. This process is further illustrated with the pair of words in (15b), where harmony is triggered by a nasal and gets suspended by a voiceless obstruent. Thus, whether a sound gets nasalized depends on whether the corresponding input is a nasal or voiceless consonant, and whether the previous sound was nasalized. In other words, the surface form of every segment depends on the corresponding segment in the input string and the immediately preceding segment in the *output* string, as illustrated in Figure 3.2(b). This is the property that makes nasal harmony in Warao, and similar iterative progressive spreading processes, left-OSL. Further examples of strictly local processes and the formal definitions of the ISL and OSL can be found in Chandlee [2014], Chandlee et al. [2015], Chandlee and Heinz [2018].

(15) **[Left Output Strictly Local (Left-OSL)]** Nasal Spreading in Warao [Osborn, 1996]

a. esoha-ya	'He pours'
nãõ-ỹã	'He saw'
nãõ-te	'He will come'
b. ho**nĩw̃ã**ku-hae	'It is a turtle'
pa**nãpanã-h̃ãẽ**	'It is a porpoise'



(a) ISL (b) Left-OSL (c) Left-subsequential

Figure 3.2: Information needed to compute functions in the subsequential boundary. The shaded dark segment in the output string depends on the corresponding segment in the input string, as well segments within the shaded light gray region.

Outside of the strictly local region, the subsequential class of functions contains long-distance processes, where the surface forms are determined by information in the word that is not within a bounded window. These maps are split into left and right subsequential, depending on whether the surface form of a sound depends on information an unbounded distance to the left or to the right. Sibilant harmony in Ineseño Chumash [Applegate, 1972, Poser, 1982, Hansson, 2010] is an example of a right subsequential function [Luo, 2017]. The data points in (16) show that all sibilants in a word agree with anteriority of the rightmost sibilant. In (16a) the past tense suffix /-waʃ/ triggers

the s→ʃ change, while in (b) the third-person suffix /-us/ triggers the ʃ→s change. The three data points in (c) stack on both suffixes to illustrate the the long-distance nature of this pattern. These data points show that the surface form of the initial sibilant in the word depends on the rightmost sibilant of the word, regardless of how far away it is. This long-distance dependency is what makes this pattern right subsequential, and distinguishes it from (14).

(16) **[Right Subsequential]** Sibilant Harmony in Ineseño Chumash [Applegate, 1972]

a. *Harmony triggered by /ʃ/*

s-kuti	'He sees'
ʃ-kuti-wa**ʃ**	'He saw'

b. *Harmony triggered by /s/*

ʃ-iʃ-ti**ʃ**i-jep	'They (two) show'
s-i**s**-ti**s**i-jep-u**s**	'They (two) show him'

c. *Long-Distance Harmony*

ʃ-api-**ʧ**^{h}o-it	'I have a stroke of good luck'
s-api-**ʦ**^{h}o-u**s**	'He has a stroke of good luck'
ʃ-api-**ʧ**^{h}o-u**ʃ**-wa**ʃ**	'He had a stroke of good luck'

Rhotic Dissimilation in Georgian [Bennet, 2013, Fallon, 1993] is an example of a left subsequential function [Payne, 2017]. The data points presented in (17) involve the /-uri/ suffix, where the r→l change takes place in the presence of another /r/. (a) shows that the suffix is pronounced as [-uri] when there is no /r/ in the root word. (b) shows that this morpheme undergoes dissimilation when there is another /r/ to the left and is instead pronounced as [-uli]. The three selected data points in (b) further show the non-local nature of this pattern; the /r/ which triggers dissimilation can be several syllables away. (c) shows that dissimilation is blocked if there is an /l/ intervening anywhere between /-uri/ and the /r/ that would otherwise trigger dissimilation. The surface form of the /-uri/ suffix therefore depends on non-local information to the left. Rhotic dissimilation, and left subsequential functions in general, schematically look like Figure 3.2(c), where the information needed to determine the surface form is found to the left; for every segment x in the input, the surface form of x is determined by x, and sounds anywhere to the left of x

(17) **[Left Subsequential]** Rhotic Dissimilation in Georgian [**?**Fallon, 1993]

a. *The -uri suffix*

kimi-uri	'chemical'
svan-uri	'Svan'

b. *Long-Distance Dissimilation*

gmi**r**-u**l**i	'heroic'
ast'**r**onomi-u**l**i	'astronomical'
g**r**amat'ik'-u**l**i	'grammatical'

c. *Dissimilation blocking by /l/*

kart**l**-uri	'Kartvelian'
par**l**ament'-uri	'parliamentary'

With respect to finite state transducers, the left (right) subsequential functions are those which can be computed by a deterministic FST which reads the input string from left-to-right (right-to-left). Left-OSL processes like nasal harmony in Warao are a subset of the left subsequential functions. The deterministic transducer which computes the Warao harmony pattern is presented in Figure 3.3. This transducer reads the string from left-to-right and outputs every input element faithfully, until it sees the first nasal sound. Once a nasal sound is found in the input, the transducer transitions to state q_1 where all vowels (denoted V) and voiced consonants (denoted C) become nasalized. For simplicity, the V:$\tilde{\text{V}}$ transition in state q_1 encompasses all the individual changes a:ã, o:õ, and e:ẽ, and similarly for C:$\tilde{\text{C}}$. If a voiceless consonant (denoted Ç) is encountered in the input string, the transducer transitions back to state q_0 and nasalization no longer takes place, until another nasal sound is found.

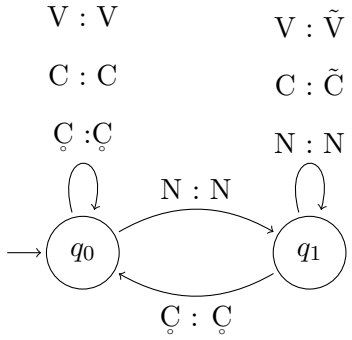


Figure 3.3: A deterministic transducer that reads input string left-to-right and computes nasal spreading in Warao. The variables N, V, C, and Ç represent nasal, vowels, voiced consonants, and voiceless consonants, respectively.

In contrast to Warao nasal harmony, the sibilant harmony pattern in Chumash is *right* subsequential. That is, the surface form of every element in the input string depends on information found to the *right*. The deterministic transducer which computes Chumash harmony, presented in Figure 3.4, must therefore read the string right-to-left. Consider for example the underlying form /s-api-ʧ^{h}o-us-wa**ʃ**/ 'He had a stroke of good luck'. When reading the string right-to-left, the transducers encounters the rightmost /ʃ/ first and transitions into state q_2 where all instances of

/s/ in the underlying form will change to [ʃ]. If the first sibilant it encounters is an /s/ instead, then it will transition to state q_1 where all instances of /ʃ/ will change to [s]. Because the transducer reads the input string from the right, the first sibilant the transducer encounters determines how all other sibilants in the word will surface. Crucially, this process could not be computed by a deterministic FST which reads the string from left-to-right because it cannot know how to output any of the sibilants in the word until it reads the end of the string.

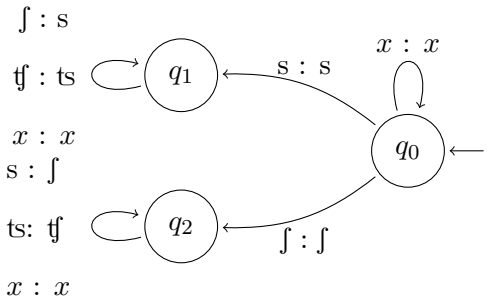
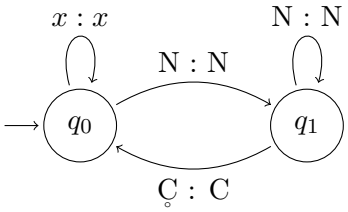
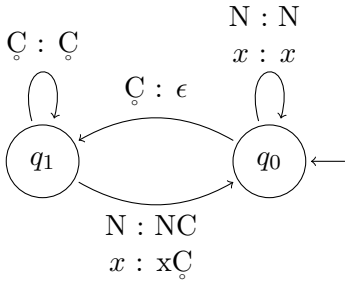


Figure 3.4: A deterministic transducer that reads the input right-to-left and computes sibilant harmony in Inseño Chumash. The variable x represents any non-sibilant sound in a string.

The ISL functions are contained within both the left and right subsequential regions of the hierarchy (Figure 3.1). These maps can be computed by a machine that reads the string left-to-right as well as right-to-left. The two contradirectional transducers which compute postnasal voicing in Zoque from (14) are presented in Figure 3.5. The right-to-left transducer outputs an empty string (ϵ)when it reads a voiceless consonant in order to wait and see whether the next sound is a nasal.



(a) String read from left-to-right. The variable x represents any non-nasal sound. The C̥:C transition indicates a voiceless obstruent becoming voiced.



(b) String read from right-to-left. The variable x represents non-nasals and non-voiceless obstruents. λ is the empty string.

Figure 3.5: Deterministic transducers that computes postnasal voicing. Variables N and C̥ represent nasals and voiceless obstruents.

3.1.2 Beyond the Subsequential Boundary

While harmony, dissimilation, and metathesis patterns across languages have been argued to be subsequential, many phonological maps do exist outside of the subsequential (deterministic) boundary. There are attested non-subsequential processes within vowel harmony [McCollum et al., 2020, Meinhardt et al., 2024, McCollum and Essegbey, 2018, Heinz and Lai, 2013], consonant harmony [Luo, 2017, Jardine and Oakden, 2023, Payne, 2017], stress [Koser, 2022, Koser and Jardine, 2020, Hao and Andersson, 2019], and tone [Jardine, 2016a]. All the functions outside of the subsequential class can be expressed as the composition of a left and right subsequential function [Elgot and Mezei, 1965]. These processes are further refined into two disjoint classes: 'unbounded circumambient' [Jardine, 2016a, McCollum et al., 2020] and 'weakly deterministic' [Heinz and Lai, 2013, Meinhardt et al., 2024, 2021]. The weakly deterministic class of functions is a properly subregular class that carves out a space between subsequential and regular [Lamont et al., 2019, Heinz and Lai, 2013].

Stem-controlled harmony patterns are weakly deterministic [Heinz and Lai, 2013]. An example of stem-controlled ATR harmony from Akan [Dolphyne, 1988, Clements, 1981, 1985, Schachter and Fromkin, 1968, Baković, 2003] is presented in (18). These data points show that the vowels in the root word determine the surface forms of the vowels in the prefixes and suffix[2]. The data points in (a) show that when the root word has [-ATR] vowels, all the vowels in the prefixes and suffixes also surface as [-ATR]. The data in (b) show that when the root word has [+ATR] vowels, all the vowels surface as [+ATR]. The contrast between the words [**ɔ**-b**ɛ**-tʊ̲-**ɪ**] and [**o**-b**e**-tu̲-**i**] highlights this pattern. The first word has the root /tʊ/ 'throw' which has a [-ATR] vowel. In this case, the third person singular (3S) prefix has the [-ATR] surface form [ɔ] and the third person singular object (3S.OBJ) suffix has the [-ATR] surface form [ɪ]. The second word has the root /tu/ 'dig' which has a [+ATR] vowel. In this case, the 3S prefix surfaces as the [+ATR] vowel [o] and the 3S.OBJ suffix surfaces as the [+ATR] vowel [i].

(18) **[Weakly Deterministic]** Stem-Controlled ATR Harmony in Akan [Clements, 1985]

a. *[-ATR] Root*		b. *[+ATR] Root*	
ɔ-tsɪ̲rɛ̲-**ɪ**	'3S-show-3S.OBJ'	**o**-fi̲ti̲-**i**	'3S-pierce-3S.OBJ'
ɔ-b**ɛ**-tʊ̲-**ɪ**	'3S-FUT-throw-3S.OBJ'	**o**-b**e**-tu̲-**i**	'3S-FUT-dig-3S.OBJ

[2]For reference, pairs of vowels that differ in ATR are presented below:

Neither a left nor right subsequential function on its own can capture this pattern; the surface form of vowels in the prefixes depends on information to the right while the surface form of vowels in the suffix depends on information to the left. Stem-controlled harmony is schematically presented in Figure 3.6. The information needed to compute the pattern is to the left of some input that undergoes harmony and to the right of another. This pattern is the composition two contradirectional subsequential functions: a left subsequential function which uses information in the root to determine the surface forms of prefixes and a right subsequential function which uses information in the root to determine surface forms of suffixes. A crucial observation here is that although the overall bidirectional harmony pattern cannot be expressed as the output of a single subsequential function, the input-output relation between *individual segments can*.

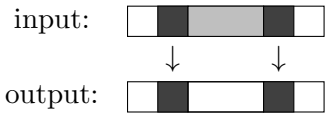


Figure 3.6: Information needed to compute stem-controlled harmony. The surface form of the first dark shaded region depends on information to the right, while the second dark shaded region depends on information to the left.

Payne [2017] presents liquid dissimilation in Yidiny [**?**Dixon, 1977] as the only exception to the generalization that dissimilation patterns are subsequential. Selected data points involving the 'going' morphemes /-ːli/ and /-ŋali/ are presented in (19) to illustrate this patterns. The data points in (a) show that the 'going' morphemes surface as [-ːr**i**] and [-ŋa**r**i] whenever there is an /l/ to right. In the data points in (b), however, this process is blocked by the /r/ in the root word on the left. This pattern is similar to rhotic dissimilation in Georgian in (17); in both patterns, dissimilation takes place except in the presence of a blocker, and both the trigger and blocker are non-local. The important distinction between them is that in Georgian, both the trigger and blocker appear to the left of the sound undergoing dissimilation, while in Yidiny, the trigger and blocker appear on opposite ends of the target of dissimilation. Liquid dissimilation in Yidiny is outside the subsequential boundary because surface forms depend on non-local information both to the left and to the right. Similar to the stem-controlled harmony pattern in Akan, liquid dissimilation in Yidiny can be expressed as the composition of two subsequential processes: a right subsequential

[+ATR]	i	e	u	o
[−ATR]	ɪ	ɛ	ʊ	ɔ

dissimilation process which makes the l→r change when there is another /l/ to the right, and a left subsequential dissimilation process which makes the r→l change when there is an /r/ to the left.

(19) **[Unbounded Circumambient]** Liquid Dissimilation in Yidiny [Dixon, 1977]

a. *Dissimilation*

magi-ːli-ɲu	'went climbing up'
magi-ːri-ŋaːl̲	'went climbing with'
duŋa-ŋali-ɲu	'went running'
duŋ-ŋari-ŋaːl̲	'went running with'

b. *Dissimilation blocking*

bur̲wa-ːli-ŋaːl̲	'went jumping with'
bur̲g-ːli-ŋaːl̲	'went walkbout with'

While ATR harmony in (18) and liquid dissimilation in (19) are both non-subsequential, Figures 3.6 and 3.7 illustrate the difference between the two patterns. The surface form of the 'going' affix in Yidiny is determined by information to the left *and* to the right of the affix in the input. This relationship between underlying and surface forms is described by Jardine [2016a] as 'unbounded circumambient' (Definition 3.2). In stem-controlled harmony, the surface form of each vowel in a word is determined by information either to the left *or* to the right. Meinhardt et al. [2024] define 'unbounded semiambient processes' (Definition 3.3) to highlight this distinction between unbounded circumambient and weakly deterministic patterns.

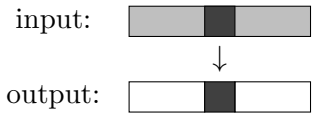


Figure 3.7: Information needed to compute unbounded circumambient patterns. Since the surface form of every segment depends on information an unbounded distance in both direction, the entire string is necessary to determine the surface form of each sound in the input.

Definition 3.2. [Jardine, 2016a] An *unbounded circumambient process* is such that

(a) its application is dependent on information (i.e., the presence of a trigger or blocker) on both sides of the target, and

(b) on both sides, there is no bound on how far this information may be from the target

Definition 3.3. [Meinhardt et al., 2024] An *unbounded semiambient process* is such that

(a) the output of any given input symbol (=target) is determined by information from at most one side of the target, and

(b) on both sides, there is no bound on how far this information may be from the target

Weakly deterministic and unbounded circumambient maps challenge the Subsequential Hypothesis because both are attested across languages. Unbounded circumambient patterns, however, are mainly attested within tonal patterns [Jardine, 2016a] and are rare within harmony patterns. The Weakly Determinstic Hypothesis [Heinz and Lai, 2013, Heinz, 2018] proposes a less restrictive bound on natural language phonology by asserting that phonological maps are weakly deterministic. The distinction between unbounded circumambient and weakly determinsitic functions is discussed further by Heinz and Lai [2013], Meinhardt et al. [2024], McCollum et al. [2018], and revisited from a logical perspective in Chapter 6 of this dissertation.

3.2 Phonology and Logic

As mentioned in Section 3.1, the regular bound on phonological grammar applies to both the stringsets used to express phonotactics and the string functions used to express phonological maps. Regular transformations are computable by non-deterministic finite state transducers, while regular stringsets are computable by finite state *acceptors*. The latter also has a *logical* characterization. The regular class of stringsets are expressible with Monadic Second Order (MSO) Logic [Buchi, 1960]. Section 3.1 alluded to a subregular hierarchy for stringsets; each of the classes within this hierarchy have logical characterizations. The tier-based strictly local class [Heinz et al., 2011] is characterized by First Order Logic, and the strictly local class of stringsets is characterized by conjunctions of negative literals. The latter class refers to stringsets which are expressed by constraints on adjacent segments. Consider for example the postnasal voicing process in Zoque that was presented in (14). The process is due to the strictly local constraint *N$\underset{\circ}{\text{C}}$ Pater [1996, 1999] which forbids sequences of sounds where a nasal is immediately followed by a voiceless consonant. A more in-depth discussion of subregular classes of stringsets and their logical characterizations can be found in [Heinz, 2018].

This section discusses the logical characterizations of string *maps* using *logical transductions*. The logical characterization of the regular stringsets was extended to MSO-definable regular trans-

ductions by Engelfriet and Hoogeboom [2001]. Logical characterizations of subregular transformations have been given by [Bhaskar et al., 2023, 2020]. These characterization are given within the formalism of Boolean Monadic Recursive Schemes, which is introduced in Section 2.2.

3.2.1 From Finite State to Logical Transducers

This section revisits the FSTs that were presented in Section 3.1.1 to show how they can be expressed as logical transducers within the BMRS formalism. The main idea is that any string transduction described by an FST can be equivalently expressed within the language of BMRS. In particular, given an FST with states $\{q_o, \ldots, q_n\}$, we can define a logical transducer (BMRS program) with recursive Boolean-valued functions $\{q_o, \ldots, q_n\}$ such that

$$q_i(x) = \top \text{ iff the machine is in state } q_i \text{ after reading the index } x \text{ of the input string}$$

Consider again the FST for Warao nasal harmony from Figure 3.3, repeated below. A sample run through the FST over the input string /naote/ is presented in Figure 3.8.

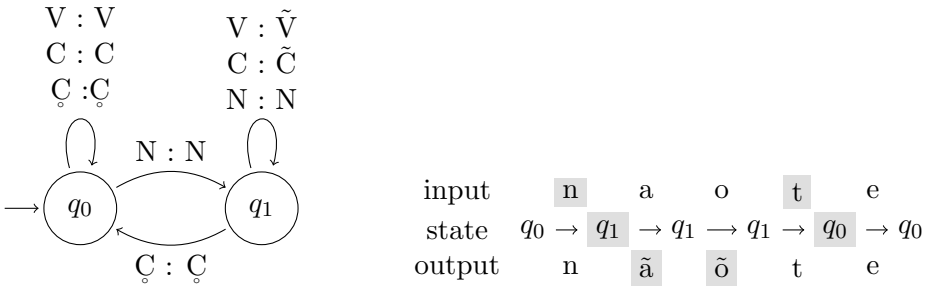


Figure 3.8: Sample computation of the string transduction /naote/→[nãõte] through the Warao nasal harmony FST.

The FST which computes Warao nasal harmony can be translated into a *logical* transducer with Boolean predicates $q_0(x)$ and $q_1(x)$ such that these predicates encode all the transition information in the original transducer. The state q_1 encodes the following information: the previous input element was either a nasal input or a nasalized output. Thus, q_1 is the state in which nasal harmony is triggered. Similarly, q_0 is the state in which nothing gets nasalized, either because a nasal input has not be observed, or because a voiceless consonant blocked nasalization. The corresponding logical transducer which encodes this information through a logical representation is presented in (20). Consider again the state q_1; the conditions under which the machine will transition into this

state after reading some input symbol are given by the `if...then...else` expression for $q_1(x)$. If the transducer reads a nasal input then the machine will transition into q_1. In other words, $q_1(x)$ will evaluate to $\top$. Otherwise, if the input symbol is either the initial element in the string or a voiceless consonant, then the machine will *not* transition into q_1. In this case, $q_1(x)$ will evaluate to $\bot$. In all remaining cases, the machine will be in state q_1 if and only if it was in q_1 after reading the previous input symbol (i.e. $q_1(px)$ holds). The logical expression in $q_1(x)$ therefore captures all the conditions under which the transducer will transition into state q_1. A similar logical translation is given in $q_0(x)$ to capture the condition under which the transducer will transition into state q_0.

(20) *Warao nasal harmony FST (Figure 3.8) expressed as a logical transducer*

$$
\begin{aligned}
q_0(x) \quad &= \quad \text{if } N(x) \text{ then } \bot \\
&\qquad \text{else if } \mathring{\text{C}}(x) \text{ then } \top \\
&\qquad \text{else } (\textsc{initial}(x) \vee q_0(px)) \\
q_1(x) \quad &= \quad \text{if } N(x) \text{ then } \top \\
&\qquad \text{else if } (\textsc{initial}(x) \vee \mathring{\text{C}}(x)) \text{ then } \bot \\
&\qquad \text{else } q_1(px) \\
[\text{nasal}]'(x) \quad &= \quad q_1(x)
\end{aligned}
$$

The final predicate in (20) gives the conditions under which an input symbol will be [+nasal] in the *output*. Since q_1 is the state in which symbols output as [+nasal], and q_0 is the state in which symbols output as [-nasal], whether or not the symbol at index x will have the feature [nasal] in the output is equivalent to the conditions under which the transducer will transition into state q_1 after reading the input at index x. In other words, $[\text{nasal}]'(x) = \top$ if and only if $q_1(x) = \top$. A demonstration of the program in (20) over the input string /naote/ is presented in Figure 3.9. This table shows how the BMRS program emulates exactly the same computation as Figure 3.8.

input	n	a	o	t	e
	N	V	V	C̊	V
$q_0(x)$	$\bot$	$\bot$	$\bot$	$\top$	$\top$
$q_1(x)$	$\top$	$\top$	$\top$	$\bot$	$\bot$
$[\text{nasal}]'(x)$	$\top$	$\top$	$\top$	$\bot$	$\bot$
state	q_1	q_1	q_1	q_0	q_0
output	n	ã	õ	t	e

Figure 3.9: Computation of the string transduction /naote/→[nãõte] through the logical transducer in (20). The functions $\{q_0, q_1\}$ and the corresponding outputs emulate exactly the same computation as the FST in Figure 3.8.

Since the $[\text{nasal}]'$ output predicate only depends on $q_1(x)$, the logical transduction in (20) can be simplified further. The recursive equation in (21) computes exactly the same logical transduction

as (20). Occurrences of N and C̬ has been replaced by [nasal] and (¬[son]∧¬[voice]) respectively, so that the final expression only makes reference to phonological features. This equation is revisited in Section 3.3 in a discussion of licensing and blocking structures.

(21) *Simplified logical transducer which does not use q_0 and q_1*

$$\begin{aligned}[\text{nasal}]'(x) = \quad & \text{if } [\text{nasal}](x) \text{ then } \top \\ & \text{else if } (\neg[\text{son}](x) \wedge \neg[\text{voice}](x)) \text{ then } \bot \\ & \text{else } [\text{nasal}]'(px)\end{aligned}$$

This demonstration gives us three ways of thinking about the Warao nasal harmony pattern. In terms of the kind of information needed to determine whether a sound in the surface form is nasalized, we only need *information to the left*. In terms of computation, the string transformation can be computed by a *deterministic FST which reads the string from left-to-right*. In terms of logic, the string transformation can be represented by a *recursive logical transducer which only uses the predecessor function*. These three ways of thinking about the pattern capture what it means for a function to be *left subsequential*.

A similar observation can be made for the right subsequential functions. Consider again the transducer which computes the sibilant harmony pattern in Inseño Chumash, repeated below with a sample string transduction in Figure 3.10.

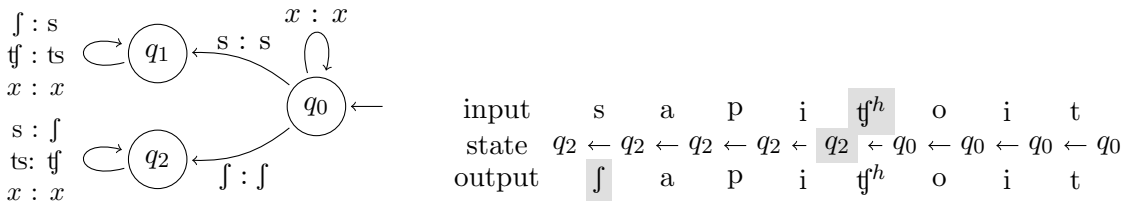


Figure 3.10: Sample computation of the string transduction /sapiʧʰoit/→[ʃapiʧʰoit] through the Chumash sibilant harmony FST.

Because the information which determines the surface form of sibilants is found to the right, the transducer reads the string from right-to-left. This FST can be translated to the BMRS program given in (22) which has recursive functions $\{q_0, q_1, q_2\}$ for each of the states in the FST. A notable contrast between the programs in (20) and (22) is that the equations in (20) use the predecessor function while the equations in (22) use the successor function.

(22) *Chumash sibilant harmony FST (Figure 3.10) expressed as a logical transducer*

$$
\begin{aligned}
q_0(x) &= \text{if } [stri](x) \text{ then } \bot \\
&\quad \text{else if } \texttt{final}(x) \text{ then } \top \\
&\quad \text{else } (q_0(sx)) \\
q_1(x) &= \text{if } \texttt{final}(x) \text{ then } ([stri](x) \wedge [ant](x)) \\
&\quad \text{else if } (q_0(sx) \text{ then } ([stri](x) \wedge [ant](x)) \\
&\quad \text{else } q_1(sx) \\
q_2(x) &= \text{if } \texttt{final}(x) \text{ then } ([stri](x) \wedge \neg[ant](x)) \\
&\quad \text{else if } (q_0(sx) \text{ then } ([stri](x) \wedge \neg[ant](x)) \\
&\quad \text{else } q_2(sx) \\
[\text{ant}]'(x) &= \text{if } [\text{stri}](x) \text{ then } q_1(x) \\
&\quad \text{else } [\text{ant}](x)
\end{aligned}
$$

Consider first the state q_0; the transducer is in this state as long as it has not encountered any sibilants in the input string. If the transducer encounters a sibilant at index x, then it will transition out of q_0; in this case $q_0(x)$ evaluates to $\bot$, given in the first line of the equation. Otherwise, if x is the final index of the string, then the transducer will remain in state q_0 since it hasn't seen any sibilants yet; in this case $q_0(x)$ evaluates to $\top$. If x is neither the final index of the input string nor a sibilant, then the machine will transition into q_0 if it was in q_0 after reading the previous input symbol. However, because the string is being read from right-to-left, the previous input symbol is found at index $x + 1$. In other words, the machine stays in state q_0 after reading the symbol at index x if it was in state q_0 after reading the symbol at index $x + 1$. For this reason, the final line of the expression for $q_0(x)$ evaluates to $q_0(sx)$. This highlights the intuition for why left subsequential transducers are expressed with programs that use the predecessor function while right subsequential transducers are expressed with programs that use the *successor* function.

The equations for $q_1(x)$ and $q_2(x)$ encode information about whether the rightmost sibilant from index x is [+ant] or [-ant]. Consider $q_1(x)$, for example. If x is the final index of the input string, then the machine moves into state q_1 if the symbol at x is a [+ant] sibilant. The equation uses the feature [strident] for sibilants. If the machine was previously at state q_0, then it will transition into q_1 under the same condition. Otherwise, the machine will be in q_1 if and only if it has already transitioned into q_1; the recursive call in the last line of $q_1(x)$ expresses this condition. As a result, $q_1(x) = \top$ if the rightmost sibilant from index x is [+ant]. Similarly, $q_2(x) = \top$ if the rightmost sibilant from index x is [-ant]. The final equation in (22) gives the condition under which

the symbol at index x has the feature [anterior] in the output; if x carries a sibilant in the input, then it will output as [+ant] if the machine is in state q_1 after reading x; otherwise, it will output as [+ant] if and only if it is [+ant] in the input.

A sample computation of the logical transducer in (22) is presented in Figure 3.11 over the input string /sapitʧʰoit/. For simplicity, the input predicate [ant] is only presented for the sibilants of the word. The predicate $q_0(x)$ evaluates to $\bot$ at index 4 because ʧʰ is a sibilant, and evaluates to $\top$ at index 7 because it is a non-sibilant and the final index of the string. The evaluation of q_0 at the remaining indices is determined by $q_0(sx)$. The remaining truth values for $q_1(x)$ and $q_2(x)$ are determined by q_0. The logical transducer emulates exactly the same transduction as the FST for sibilant harmony; it is in state q_0 at the end of the string and moves out of q_0 into q_2 when it reads the first sibilant at index 4.

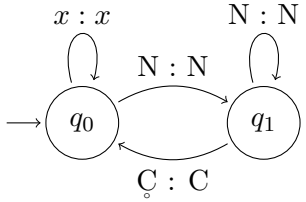
input	s	a	p	i	ʧʰ	o	i	t
index	0	1	2	3	4	5	6	7
[ant](x)	$\top$				$\bot$			
$q_0(x)$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\top$	$\top$	$\top$
$q_1(x)$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
$q_2(x)$	$\top$	$\top$	$\top$	$\top$	$\top$	$\bot$	$\bot$	$\bot$
[ant]$'(x)$	$\bot$				$\bot$			
state	q_2	q_2	q_2	q_2	q_2	q_o	q_o	q_o
output	ʃ	a	p	i	ʧʰ	o	i	t

Figure 3.11: Computation of the string transduction /sapitʃoit/→[ʃapitʃoit] through the logical transducer in (22). The functions $\{q_0, q_1, q_2\}$ and the corresponding outputs emulate exactly the same computation as the FST in Figure 3.10.

The definition for the output predicate [ant]$'(x)$ depends on the recursive function $q_1(x)$, which encodes information about the anteriority of the rightmost sibilant in the string. In contrast to the Warao nasal harmony example in (20), this expression does not simplify to something that doesn't make reference to q_1. In the case of Warao harmony, we only need to know whether the following index is nasalized in the output. Thus, [nasal]$'(x)$ is defined in terms of [nasal]$'(px)$. In the case of sibilant harmony, the long-distance nature of the pattern means that we do not know in advance which index carries the relevant information for determining whether a segment will be [anterior] in the input. In other words, [ant]$'(x)$ cannot be defined in terms of [ant]$'(s^n x)$ for any particular n. Instead, we need a separate recursive function that traverses the entire string to determine whether the rightmost sibilant is [+ant]; this is precisely what the $q_1(x)$ function accomplishes. This distinction was discussed in Section 2.2 with the two functions $f_{it} : a \to b/b\_(iterate)$ in

(9) and $f^* : a \to b/bx^*\_\_$ in (12). The OSL function f_{it} is similar to Warao nasal harmony in (21), where the output predicates in the corresponding logical transductions are defined in terms of both input and output predicates. The subsequential function f^* is similar to Chumash sibilant harmony in (22), where the output predicates in the corresponding logical transductions require an auxiliary recursive predicate `b-left`. The difference between these two functions highlights the distinction between OSL and subsequential maps from a logical perspective.

The ISL functions are at the intersection of left and right subsequential; it was shown in Figure 3.5 that the ISL map for postnasal voicing can be modeled with an FST that reads the string left-to-right as well as right-to-left. The left-to-right transducer that computes postnasal voicing is repeated below in Figure 3.12 with a sample computation of /mpama/→[mbama]. The logical transducer for this FST is given in (23). The machine transitions to state q_1 if and only if it reads a nasal input (i.e. $q_0(x) = N(x)$), and transitions to state q_0 whenever it reads a non-nasal (i.e. $q_0(x) = \neg N(x)$). Figure 3.13 presents a sample computation of this logical transducer.



input	m	p	a	m	a	
state	$q_0 \to$	q_1	$\to q_0 \longrightarrow q_0 \to$	q_1	$\to q_1$	
output	m	b	a	m	a	

Figure 3.12: Sample computation of the string transduction /mpama/→[mbama] through the left-to-right postnasal voicing FST.

(23) *Postnasal voicing (Figure 3.12) expressed as a logical transducer*

$$\begin{aligned} q_0(x) &= \neg N(x) \\ q_1(x) &= N(x) \\ [\text{voi}]'(x) &= \text{if } \big(\mathring{\text{C}}(x) \wedge q_1(px)\big) \text{ then } \top \\ &\quad \text{else } [\text{voi}](x) \end{aligned}$$

The notable difference between the BMRS program for postnasal voicing in (23) and the programs for nasal and sibilant harmony in (20) and (22) is that the program in (23) is *not recursive*. In fact, this program can be simplified further because the definition of $[\text{voi}]'(x)$ depends only on q_1 which simply just evaluates to $N(x)$. Thus, the three equations in (23) can be simplified to the single equation in (24). This program is similar to the ISL function $f : a \to b/b\_\_$ from (6).

input	m	p	a	m	a
$\mathring{\text{C}}(x)$	$\bot$	$\top$	$\bot$	$\bot$	$\bot$
$q_0(x)$	$\bot$	$\top$	$\top$	$\bot$	$\top$
$q_1(x)$	$\top$	$\bot$	$\bot$	$\top$	$\bot$
$[\text{voi}]'(x)$	$\top$	$\top$	$\top$	$\top$	$\top$
state	q_1	q_0	q_0	q_1	q_1
output	m	b	a	m	a

Figure 3.13: Computation of the string transduction /mpama/→[mbama] through the logical transducer in (23). The functions $\{q_0, q_1\}$ and the corresponding outputs emulate exactly the same computation as the FST in Figure 3.12.

(24) *Simplified logical transduction for postnasal voicing*

$$[\text{voi}]'(x) = \text{if } \big(\neg[\text{son}](x) \wedge \neg[\text{voi}](x) \wedge [\text{nas}](px)\big) \text{ then } \top$$
$$\text{else } [\text{voi}](x)$$

The examples of ISL, OSL, and subsequential functions presented in this section give a logical perspective on the different classes of functions within the subsequential boundary. A logical perspective on the distinction between weakly deterministic and unbounded circumambient maps is discussed in Chapter 6.

3.2.2 Logical Characterizations of String Functions

The fundamental result connecting the discussion in Section 3.1 with the BMRS formalism in Section 2.2 is that BMRS programs model exactly the class of rational string functions [Bhaskar et al., 2023], expressed below as Theorem 3.4. More specifically, what this theorem states is that the well-defined and order-preserving BMRS programs over monadic string models characterize exactly the rational string functions. The term 'well-defined' means that for every input string model, the resulting structure obtained from the BMRS program is also a string model (Definition 2.3 in Section 2.1). Since we are mainly concerned with feature models in this dissertation, well-definedness takes on a different meaning which we will not explore in-depth. The term 'order-preserving' means that for every index x in the input model, all copies of index x in the output model precede copies of index $x + 1$ (Definition 2.11 in Section 2.2.3). In the characterization results presented here, 'BMRS transductions' refers to well-defined and order-preserving BMRS programs over monadic string models.

Theorem 3.4 (**Logical characterization of rational functions**)**.** [Bhaskar, Chandlee, and Jardine, 2023] BMRS transductions are equivalent to the rational transductions.

The class of all BMRS programs is denoted $\text{BMRS}^{p,s}$, to indicate that it is the class of programs in which recursive equations are defined over a signature with both the predecessor and successor functions. Subregular classes of functions are characterized in terms of restrictions on $\text{BMRS}^{p,s}$. BMRS^{p}, for example, is the restricted class of programs which does not contain any terms of the form $s(T)$. Section 3.1.1 gave examples of cross-linguistics phonological patterns which fall within the subsequential boundary. Sibilant harmony in Inseño Chumash in (16) was presented as an example of a *right* subsequential function. From the perspective of linguistics, the property that makes this map right subsequential is that the surface form of any sibilant in the underlying form depends on information an unbounded distance to the right (i.e. the rightmost sibilant in the word). In terms of FSTs, this map is computed by a deterministic transducer that reads the string *right-to-left* (Figure 3.4) because the rightmost sibilant must be read first in order to determine how all subsequent sibilants will output. The BMRS program which models exactly the same transduction was presented in (22), where it was noted that only the successor function is necessary because each index x in the input string only needs information from indices to the right of x. This observation is precisely the property of BMRS transductions that characterize the right subsequential maps (and similarly for left subsequential). The logical characterizations of the subsequential functions are summarized in Theorem 3.5.

Theorem 3.5 (**Logical characterization of subsequential functions**)**.** [Bhaskar, Chandlee, Jardine, and Oakden, 2020] BMRS transductions which only use the *predecessor* function (BMRS^{p}) are equivalent to the *left-subsequential* functions. BMRS transductions which only use the *successor* function (BMRS^{s}) are equivalent to the *right subsequential* functions.

Postnasal voicing in (14) was presented as an example of an input strictly local process. The BMRS transduction which models this function was given in (24). Unlike the examples of harmony in Chumash and Warao, the logical transduction for postnasal voicing did not make use of recursion. Intuitively, this is because ISL functions have a fixed window within which the necessary information can be found, and therefore only input predicates and a fixed number of nested occurrences of successor and predecessor are necessary. Chandlee and Lindell [forth.] give a logical characterization of a subclass of ISL functions as quantifier-free interpretations over monadic string models. These quantifier-free interpretations are essentially the non-recursive programs we define in BMRS, thus

yielding the characterization in Theorem 3.6. This definition refers to 'finite-to-one' functions; a function $f : X \to Y$ is finite-to-one iff for every $y \in Y$, $\{x \in X \mid f(x) = y\}$ is finite. The phonological maps which are discussed in this dissertation are all finite-to-one. Thus we do not go further in-depth with this characterization. The logical characterizations discussed in this section are summarized in Figure 3.14. Figure 3.14 is revisited in Chapter 6 with the addition of a BMRS characterization of the weakly deterministic functions. Logical characterizations of OSL functions remains an open problem.

Theorem 3.6 (**Logical characterization of ISL functions**). [Chandlee and Lindell, forth.] Non-recursive BMRS transductions (NR-BMRS^{p,s}) are equivalent to finite-to-one transductions.

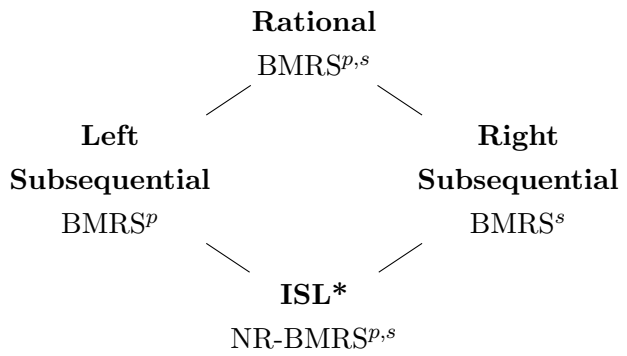


Figure 3.14: Logical characterizations of rational, subsequential, and (*finite-to-one) ISL functions with respect to BMRS programs [Bhaskar et al., 2020, 2023, Chandlee and Lindell, forth.].

3.3 Discussion

The discovery that SPE rewrite rules express rational transductions made it possible to adapt existing work in formal language theory (FLT) to analyze the complexity of phonological maps. The use of FSTs in computational phonology has led to testable hypotheses about the complexity of phonological maps [Heinz, 2018, Chandlee and Heinz, 2018, Payne, 2017, Luo, 2017, Heinz and Lai, 2013, Gainor et al., 2012, Chandlee et al., 2012], and to learning algorithms for classes of phonological maps [Chandlee, 2014, Jardine et al., 2014, Heinz, 2009, 2010b]. On the other hand, Pater [2018] argues that while transducers are valuable in *describing* the complexity of patterns, they are not a good framework of phonological theory as they do not provide a way to encode phonological *generalizations*, and FLT therefore cannot be an alternative to constraint-based framework of Optimality Theory (OT).

> FLT allows us to make general statements about the formal power needed to capture different aspects of natural language, in a way that is orthogonal to distinctions between various rule-based and constraint-based theories. [...] While there are good reasons to use FLT in the phonological enterprise, I think it is a mistake to conceive of FLT as an alternative to OT (and other theories), and this reply points out one reason why: it provides no obvious way of stating the kinds of substantive restrictions on phonological systems that are needed to delimit phonological typology. [Pater, 2018, p.156]

The appeal of BMRS as a framework of phonology is in its combination of restrictiveness with its ability to capture phonological generalizations. Chandlee and Jardine [2021] argue that the BMRS formalism 'has a well-understood complexity bound that corresponds to previous results in the study of computational phonology, but it also provides a way to implement phonological substance' [Chandlee and Jardine, 2021, p.3]. They show, for example, that the `if...then...else` syntax of BMRS programs yields a very natural representation of licensing and blocking structures. Consider the schematic program $F'(x)$ in (25), where $\texttt{STRUCT}_i(x)$ is any Boolean-valued BMRS expression. Programs of this form can be used to model licensing and blocking in the following way. In any line of the form [`if` $STRUCT_i(x)$ `then` $\top$], $\texttt{STRUCT}_i(x)$ is a *licensing* structure; whenever $\texttt{STRUCT}_i$ holds, the output at index i will have property/feature F. Similarly, any line of the form [`if` $STRUCT_i(x)$ `then` $\bot$], $\texttt{STRUCT}_i(x)$ is a *blocking* structure; whenever $\texttt{STRUCT}_i$ holds, the output at index i will *not* have property/feature F.

(25) *BMRS equations with licensing and blocking structures* [Chandlee and Jardine, 2021]

$$
\begin{aligned}
F'(x) = \quad & \text{if } \texttt{STRUCT}_1(x) \text{ then } \{\top, \bot\} \text{ else} \\
& \text{if } \texttt{STRUCT}_2(x) \text{ then } \{\top, \bot\} \text{ else} \\
& \vdots \\
& \text{if } \texttt{STRUCT}_n \text{ then } \{\top, \bot\} \text{ else} \\
& F(x)
\end{aligned}
$$

Chandlee and Jardine [2021] show how these structures can be used to represent phonological generalizations, including the typology of *NC̥ effects. In a constraint-based framework, *NC̥ expresses a constraint against sequences of sounds where a nasal is immediately followed by a voiceless consonant. This constraint explains a range of processes found across languages, which can be explained in terms of each language's relative ordering of *NC̥ with other constraints Pater [1996, 1999]. Within BMRS, it is possible to give a Boolean-valued expression which return $\top$ iff an index violates the constraint, presented in (26). (26a) evaluates to $\top$ for any nasal segment that

is *followed by* a voiceless consonant, and (26b) evaluates to $\top$ for any voiceless consonant that is *preceded by* a nasal. A few brief examples are presented here to show how the logical implementation of the NC̥ constraint in (26) and the license and blocking structures in (25) can be used to model various NC̥ effects across languages.

(26) a. $\mathbf{N\mathring{C}}(x) = \text{if } [nas](x) \text{ then } \big(\neg[son](s(x)) \wedge \neg[voi](s(x))\big) \text{ else } \bot$

b. $\mathrm{N}\mathbf{\mathring{C}}(x) = \text{if } \big(\neg[son](x) \wedge \neg[voi](x)\big) \text{ then } [nas](p(x)) \text{ else } \bot$

Languages employ various phonological transformations to ensure that surface forms do not have adjacent nasal and voiceless consonant sounds. One such resolution is postnasal voicing: voiceless consonants become voiced when they are immediately proceeded by a nasal sound. This pattern was observed in Zoque, presented in (14). The BMRS program which models postnasal voicing was presented in (24). This program can equivalently be expressed as (27), where N$\mathbf{\mathring{C}}$ in (26b) is a *licensing* structure for the feature [voice].

(27) *Postnasal voicing with N$\mathbf{\mathring{C}}$ as a licensing structure for* [voice]

$[voi]'(x) = \text{if } \mathrm{N}\mathbf{\mathring{C}}(\mathrm{x}) \text{ then } \top \text{ else } [voi](x)$

Languages may also resolve NC̥ through nasalization, in which the second sound in the sequence becomes [+nasal], or through denasalization in which the first sound in the sequence becomes [-nasal]. Nasalization and denasalization as resolutions for an NC̥ sequence are presented in (28) and (29), respectively. In the case of nasalization, the expression $\mathrm{N}\mathbf{\mathring{C}}(x)$ is a *licensing* structure for the feature [nasal]. In the case of denasalization, $\mathbf{N\mathring{C}}(x)$ is a *blocking* structure for the feature [nasal]. These examples show how BMRS can be used to model the relevant phonological generalizations underlying NC̥ typology effects. A more in-depth discussion and further examples that demonstrate this point can be found in Chandlee and Jardine [2021].

(28) *Nasalization with N$\mathbf{\mathring{C}}$ as a licensing structure for* [nasal]

$[nas]'(x) = \text{if } \mathrm{N}\mathbf{\mathring{C}}(\mathrm{x}) \text{ then } \top \text{ else } [nas](x)$

(29) *Denasalization with $\mathbf{N\mathring{C}}$ as a blocking structure for* [nasal]

$[nas]'(x) = \text{if } \mathbf{N\mathring{C}}(\mathrm{x}) \text{ then } \bot \text{ else } [nas](x)$

The BMRS framework brings together the merits of both FLT and OT; it is a restrictive framework that makes it possible to state and prove computational properties of phonological

maps, while also making it possible to encode phonological generalizations. These facts about BMRS are explored throughout the remainder of this dissertation. Chapters 4 and 5 discusses the latter point. Chapter 4 presents a learning procedure for ISL maps that directly makes reference to phonological generalizations underlying input-output relations. Chapter 5 shows how process interaction can be modeled naturally with BMRS programs. Chapter 6 focuses on the former point via the class of weakly deterministic functions. It is not only possible to define the subclass of programs which model weakly deterministic functions, but such a definition also lends itself well to *reasoning* about the expressivity of phonological maps.

4

LEARNING LOCAL PROCESSES WITH LOGICAL TRANSDUCTIONS

Input Strictly Local (ISL) functions are learnable from positive data. Chandlee [2014] shows that for every ISL function f it is possible to learn a finite state transducer (FST) which computes f from a characteristic sample $S \subseteq f$ of input-output pairs of strings. This chapter revisits the problem of learning ISL maps from the perspective of *logical* transducers, and presents an approach for learning a BMRS program which models an ISL function from a sample of input-output pairs of *features models*. Given the discussion on equivalences between FSTs and BMRS programs in Section 3.1, one way to approach the learning problem is to simply learn a logical transducer using the same methods as learning an FST. While this may be a sufficient solution to the problem of learning, the purpose of this chapter is to show that BMRS can be used to learn *phonological generalizations*. Section 4.1 proposes a canonical normal form for ISL BMRS program, which refines the learning problem and provides an alternative means of representing phonological generalizations from the licensing and blocking structures discussed in Section 3.3. Section 4.2 discusses previous work on phonological learning with partially ordered hypothesis spaces, and 4.3 shows how the same structures and principles can be adapted to learning transformations with BMRS. Section 4.3.2 presents the learning procedure with the postnasal voicing process in Zoque as a case study.

4.1 Input Strictly Local BMRS Programs

Input Strictly Local (ISL) processes are those in which there is window of fixed size k such that the surface form of every element in a word is dependent on information found within some k-window around the corresponding input. Postnasal voicing in Zoque was presented in Section 3.1.1 as an example of an ISL function; the input-output mapping /mpama/→[mbama] 'my clothing' is presented in Figure 4.1. This map is characterized as an ISL-2 function because the surface form of every segment x depends on information within a window of size 2 around x (specifically x and the sound immediately preceding x). Every ISL function is ISL-k for some k.

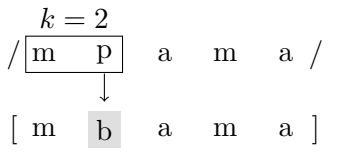


Figure 4.1: Postnasal voicing is an ISL-2 map; the [b] sound in the surface form [mbama] 'my clothing' depends on information contained within the illustrated window of size 2 around /p/.

Mathematical and formal language-theoretic characterizations of ISL functions and can be found in Chandlee [2014] and Chandlee and Heinz [2018]. With respect to logical transducers, (finite-to-one) ISL functions are exactly those that can be modeled by non-recursive BMRS programs (NR-BMRS^{p,s}) [Chandlee and Lindell, forth.]. This characterization was discussed in Section 3.2. This chapter only considers length-preserving ISL functions, which are always finite-to-one. Thus the characterization of ISL as NR-BMRS programs holds up. The purpose of this section is to *refine* this characterization further with a *syntactic* description of ISL BMRS programs which we will refer to as the 'canonical normal form'.

Regardless of the formal representation of a string function (e.g. finite state transducer or BMRS program), every string function always has more than one possible way it can be modeled. Recall the string function $f : a \to b/b\_\_$, which is modeled by the program in (6) from Section 2.2. Similar to postnasal voicing, this function is ISL-2.

(6) *The function $f : a \to b/b\_\_$ as a BMRS program*
$P_a^f(x)$ = if $P_b(px)$ then $\perp$ else $P_a(x)$
$P_b^f(x)$ = if $P_a(x)$ then $P_b(px)$ else $P_b(x)$

The equation for $P_b^f(x)$ is one of infinitely many ways to give a logical description for when an index carries 'b' in the output. All of the equations in (30) are logically equivalent ways of expressing $P_b^f(x)$. Moreover, all the equations in (30) give a logical expression of the same idea; an

index carries a 'b' in the output if it is either a 'b' in the input or it is an 'a' that is preceded by a 'b' in the input. Learning this string transformation as a logical expression amounts to learning precisely this relationship. Therefore, to approach the problem of learning this relationship, it is first necessary to determine how this relationship will be *represented*. In other words, which of the possible representations should be the output of the learning procedure?

(30) *Several logically equivalent ways of expressing* $P_b^f(x)$

$$\begin{aligned} P_b^f(x) &= \text{if } P_a(x) \text{ then } P_b(px) \text{ else } P_b(x) \\ &\equiv \text{if } P_b(px) \text{ then } P_a(x) \text{ else } P_b(x) \\ &\equiv \big(P_a(x) \wedge P_b(px)\big) \vee P_b(x) \\ &\equiv \text{if } P_b(x) \text{ then } \top \text{ else } \big(P_a(x) \wedge P_b(px)\big) \end{aligned}$$

Consider the final expression in (30). The expression checks first whether an index carries 'b' in the input. This means that the determination for whether an index will carry a 'b' in the *output* is relativized to whether it carries a 'b' in the *input*. If an index x carries a 'b', the 'then' part of the expression captures the fact that x will remain a 'b'. Otherwise, the 'else' part of the expression gives conditions for when the non-'b' x will become a 'b' in the output. The equation for $P_a^f(x)$ in (6) can also be rewritten as a logically equivalent expression in a similar form, given in (31). This final expression in (31) relativizes whether an index will carry 'a' in the output to whether it carries 'a' in the input. When an index x carries 'a' in the input, the 'then' part captures the conditions under which it will remain an 'a'. In particular, x will remain an 'a' in the output if it is not preceded by a 'b'. Otherwise, the 'else' part of the expression states that a non-'a' input will never become an 'a' in the output.

(31) *Logically equivalent ways of expressing* $P_a^f(x)$

$$\begin{aligned} P_a^f(x) &= \text{if } P_b(px) \text{ then } \bot \text{ else } P_a(x) \\ &\equiv P_a(x) \wedge \neg P_b(px) \\ &\equiv \text{if } P_a(x) \text{ then } \neg P_b(px) \text{ else } \bot \end{aligned}$$

The final expressions in (30) and (31) have a similar syntactic form, where the Boolean value for an output predicate $P_\sigma^f(x)$ is defined in terms of the input predicate $P_\sigma(x)$ in the following way: if P_σ holds of the input, the 'then' part of the equation determines the output, and if P_σ does not hold of the input, the 'else' part of the equation determines the output. Section 4.1.1 provides a

more formal description of this concept with a definition of canonical normal form (CNF), and a demonstration of how any BMRS equation can be translated to this form. The formal details of this translation can be found in the Appendix.

4.1.1 Normal Form for ISL Programs

Recall from the discussion on BMRS syntax in Section 2.2.1 that variables are terms, and for any term T, $s(T)$ and $p(T)$ are terms. Given an input signature $\Sigma = \langle P_\sigma, s, p\rangle_{\sigma\in\Sigma}$, we define the *atoms* of Σ as the set of Boolean-valued expression in (4.1).

$$atoms(\Sigma) := \{P_\sigma(T) \,|\, \sigma \in \Sigma \text{ and } T \text{ is a term}\} \tag{4.1}$$

Definition 4.1 (**Canonical Normal Form for ISL BMRS Programs**)**.** Let $\{P'_\sigma\}_{\sigma\in\Sigma}$ be a system of (non-recursive) equations over the input signature $\Sigma = \langle \rtimes, \ltimes, P_\sigma, s, p\rangle_{\sigma\in\Sigma}$. Each P'_σ is in canonical normal form (CNF) if it is expressed as in (4.2), where ϕ_σ and ψ_σ are either $\top$, $\bot$, or disjunctive normal form formulas over $atoms(\Sigma)$.

$$P'_\sigma(x) = \text{if } P_\sigma(x) \text{ then } \neg\phi_\sigma(x) \text{ else } \psi_\sigma(x) \tag{4.2}$$

A BMRS program over the signature Σ is in CNF if every equation $\{P'_\sigma\}_{\sigma\in\Sigma}$ is in CNF.

The syntactic form in (4.2) breaks the equation into two parts: ϕ_σ expresses the conditions under which something that is P_σ in the input will no longer be P_σ in the output, and ψ_σ expresses the conditions under which something that is not P_σ in the input will become P_σ in the output. One thing to note about the CNF is that it is defined in terms of disjunctive normal form (DNF) formulas. In particular for each $\sigma \in \Sigma$, the formulas ϕ_σ and ψ_σ are both disjunctions of conjunctions of expressions (or their negations) in $atoms(\Sigma)$. However, these three propositional logic operators are not part of the BMRS syntax. The definitions of conjunction, disjunction, and negation using the `if...then...else` syntax of BMRS were presented in (5) of Section 2.2.1. The formulas ϕ_σ and ψ_σ are therefore made up of nested `if...then...else` formulas with a particular structure. We use the logical operators in order to make the programs compact and readable.

The CNF of the program in (6) is presented in (32). These are exactly the syntactic forms that were discussed in (31) and (30), with the exception that the 'then' part of $P^*_b(x)$ is $\neg\bot$ rather than $\top$. ϕ_a and ψ_b are highlighted. $\phi_a(x) = P_b(px)$ expresses the conditions under which an index that

carries an 'a' in the input will not carry an 'a' in the output. Similarly, $\psi_b(x) = P_a(x) \wedge P_b(px)$ expresses the conditions under which an index that does not carry a 'b' in the input will carry a 'b' in the output. The remaining parts $\phi_b(x)$ and $\psi_a(x)$ are both $\bot$ because a 'b' in the input string never stops being a 'b' and a non-'a' in the input string never becomes an 'a'. It is easy to confirm via truth tables that P_a^* and P_b^* are logically equivalent to P_a^f and P_b^f, respectively. Theorem 4.2 states that indeed *every* ISL function can be rewritten as an equivalent one in this syntactic form.

(32) *The function $f : a \to b/b\_\_$ as a CNF BMRS program*

$$P_a^*(x) = \text{if } P_a(x) \text{ then } \neg P_b(px) \text{ else } \bot$$
$$P_b^*(x) = \text{if } P_b(x) \text{ then } \neg\bot \text{ else } \big(P_a(x) \wedge P_b(px)\big)$$

Theorem 4.2. Every (length-preserving) NR-BMRS^{p,s} program P can be expressed as a logically equivalent BMRS program P^* such that all the equations in P^* are in canonical normal form.

The proof of this theorem is presented at the end of the section. The main idea is that for any output predicate P'_σ, there is a systematic way to construct an equivalent CNF predicate P^*_σ by generating a truth table of all the atoms in the definition of P'_σ, and determining ϕ_σ and ψ_σ from that truth table. As an example, Figure 4.2 presents the truth tables generated from $\{P_a^f, P_b^f\}$ in (6). The table in (a) has columns for $P_a(x)$ and $P_b(px)$ because these are the atomic expressions that appear in the equation $P_a^f(x)$. Similarly, the table in (b) has columns for $P_b(x)$, $P_a(x)$, and $P_b(px)$ because these are the atomic expressions in $P_b^f(x)$. The highlighted rows in the two tables contain all the relevant information to construct a CNF program. The highlighted row in (a) is the only instance in which $P_a(x)$ evaluates to $\top$ and $P_a^f(x)$ evaluates to $\bot$. The content of this row is represented by the formula $\phi_a(x) = P_b(px)$. Similarly, the highlighted row in (b) is the only instance in which $P_b(x)$ evaluates to $\bot$ and $P_b^f(x)$ evaluates to $\top$. The content of this row is represented by the formula $\psi_b(x) = P_a(x) \wedge P_b(px)$. Note that the table in (b) is missing the rows corresponding with $P_a(x) = \top$ and $P_b(x) = \top$ since an index cannot carry both 'a' and 'b'. The two truth tables in Figure 4.2 contain the four pieces of information $\{\phi_a, \psi_a, \phi_b, \psi_b\}$ which uniquely identify the string function $f : a \to b/b\_\_$.

This method of translation can be extended to feature models. Consider a postnasal voicing map [-son]→[+voi]/[+nas]__. The BMRS equation which expresses when an index will be [+voice] in the output was presented as (24), and repeated in the first line of (33). This example was previously also

$P_a(x)$	$P_b(px)$	$P'_a(x)$
⊤	⊤	⊥
⊤	⊥	⊤
⊥	⊤	⊥
⊥	⊥	⊥

(a) Truth table generated from $P_a^f(x)$. The highlighted row represents the conditions under which an 'a' input becomes a non-'a' output (i.e. ϕ_a).

$P_b(x)$	$P_a(x)$	$P_b(px)$	$P'_b(x)$
⊤	⊥	⊤	⊤
⊤	⊥	⊥	⊤
⊥	⊤	⊤	⊤
⊥	⊤	⊥	⊥
⊥	⊥	⊤	⊥
⊥	⊥	⊥	⊥

(b) Truth table generated from $P_b^f(x)$. The highlighted row represents the conditions under which a non-'b' input becomes a 'b' output (i.e. ψ_b).

Figure 4.2: Truth tables generated from the equations $\{{P_a}^f, {P_b}^f\}$ in (6). These are used to construct the CNF program $\{{P_a}^*, {P_b}^*\}$ in (32).

discussed in Section 3.3 with respect to licensing and blocking structures. For simplicity, the $\text{N}\mathring{\mathbf{C}}(x)$ constraint from (26) is given here as a shortcut for the conjunction $\big(\neg[\text{son}](x) \wedge \neg[\text{voi}](x) \wedge [\text{nas}](px)\big)$ in order to simplify presentation. In the first equation for $[\text{voi}]'(x)$, $\text{N}\mathring{\mathbf{C}}(x)$ is a licensing structure for [voi]. This equation is logically equivalent to the CNF presented as the second expression in (33). The logical equivalence of these two expressions is evident from the fact that both simply express the disjunction of $\text{N}\mathring{\mathbf{C}}(x)$ and $[\text{voi}](x)$. The CNF is, however, simpler since it does not explicitly need to include $\neg[\text{voi}](x)$ in its definition; this is implicit when the expression evaluates to the 'else' part.

(33) *Logically equivalent ways to express [voi]′(x) for postnasal voicing*

$$\begin{aligned}
\text{N}\mathring{\mathbf{C}}(x) &= \neg[\text{son}](x) \wedge \neg[\text{voi}](x) \wedge [\text{nas}](p(x)) \\
[\text{voi}]'(x) &= \text{if } \text{N}\mathring{\mathbf{C}}(x) \text{ then } \top \text{ else } [\text{voi}](x) \\
&\equiv \text{if } [\text{voi}](x) \text{ then } \neg\bot \text{ else } (\neg[\text{son}](x) \wedge [\text{nas}](p(x))) \\
&\equiv \text{N}\mathring{\mathbf{C}}(x) \vee [\text{voi}](x)
\end{aligned}$$

Figure 4.3 presents the truth table that is generated from the atoms in the output predicate $[\text{voi}]'(x)$. The highlighted row contains the conditions for $\psi_{[\text{voi}]}$: when something that is [-voice] in the input becomes [+voice] in the output. The information contained in this row is precisely the definition of $\text{N}\mathring{\mathbf{C}}(x)$: $[\text{voi}](x) = \bot$, $[\text{son}](x) = \bot$, and $[\text{nas}](x) = \top$. Because there are no situations in which a [+voice] input becomes [-voice] in the output, $\phi_{[\text{voi}]}(x) = \bot$. From this, we get the CNF in (33), with $\psi_{[\text{voi}]}$ highlighted. The three different representations in (33) are discussed further in Section 4.1.2 with respect to how phonological generalizations are encoded in BMRS programs.

[voi](x)	[son](x)	[nas](px)	[voi]$'$(x)
$\top$	$\top$	$\top$	$\top$
$\top$	$\top$	$\bot$	$\top$
$\top$	$\bot$	$\top$	$\top$
$\top$	$\bot$	$\bot$	$\top$
$\bot$	$\top$	$\top$	$\bot$
$\bot$	$\top$	$\bot$	$\bot$
$\bot$	$\bot$	$\top$	$\top$
$\bot$	$\bot$	$\bot$	$\bot$

Figure 4.3: Truth table generated from the equation [voi]$'(x)$ in (33). The highlighted row represents $\psi_{[\text{voi}]}(x)$. The information in this table is sufficient to generate a CNF expression that is logically equivalent to [voi]$'(x)$.

The formal statement of CNF in Definition 4.1 specifies that ϕ_σ and ψ_σ are disjunctive normal form (DNF) expressions. The previous two examples involve a special case with no disjunctions. Consider, however, the string function $a \to b/\{b, c\}\_\_$. The CNF BMRS program which models this function is presented in (34). The expressions for $\phi_a(x)$ and $\psi_b(x)$ are highlighted. The DNF $\psi_b(x) = (P_a(x) \wedge P_b(px)) \vee (P_a(x) \wedge P_c(px))$ represents the conditions under which a non-'b' input becomes a 'b' in the output string. In this case, a disjunction is necessary because there are two situations in which this can take place.

(34) $a \to b/\{b, c\}\_\_$

$$P'_a(x) = \text{if } P_a(x) \text{ then } \neg\, (P_b(px) \vee P_c(px)) \text{ else } \bot$$

$$P'_b(x) = \text{if } P_b(x) \text{ then } \neg\bot \text{ else } (P_a(x) \wedge P_b(px)) \vee (P_a(x) \wedge P_c(px))$$

$$P'_c(x) = \text{if } P_c(x) \text{ then } \neg\bot \text{ else } \bot$$

The non-recursive BMRS programs (NR-BMRS^{p,s}) are the logical characterization of ISL functions (Theorem 3.6). However, this characterization can be further refined with a syntactic constraint; the CNF non-recursive BMRS programs (CNF-BMRS^{p,s}) compute exactly the same functions as the general class of non-recursive programs. By definition every CNF-BMRS^{p,s} program is NR-BMRS^{p,s} because the equations only make reference to predicates in the input signature. In the reverse direction, Theorem 4.2 says every NR-BMRS^{p,s} program can be rewritten as an equivalent CNF-BMRS^{p,s}. These two facts together yield Theorem 4.3, which provides a more restrictive syntactic characterization of the ISL functions.

As a final note, the examples presented so far all assume that for every predicate P_σ, the definition of the output predicate $P'_\sigma(x)$ depends on the input predicate $P_\sigma(x)$. For phonological

processes, this is generally true. However, it is possible to construct an equivalent CNF expression even when $P'_\sigma(x)$ does not depend on $P_\sigma(x)$. This is considered in the first case of the proof of Theorem 4.2.

Theorem 4.3 (**CNF Characterization of ISL Functions**)**.** The NR-BMRS^{p,s} transductions in canonical normal form (CNF-BMRS^{p,s}) are equivalent to the ISL functions.

Proof. Follows immediately from Theorem 3.6, Theorem 4.2, and the fact that CNF-BMRS^{p,s} programs are a subset of the NR-BMRS^{p,s} programs. □

Proof of Theorem 4.2. Let $\boldsymbol{P} = \{P'_\sigma\}_{\sigma\in\Sigma}$ be a non-recursive BMRS program over the input signature $\Sigma = \{P_\sigma, p, s\}_{\sigma\in\Sigma}$. Fix $\sigma \in \Sigma$, and let $atoms(P'_\sigma(x))$ be the set of Boolean expressions in $atoms(\Sigma)$ which appear in the equation $P'_\sigma(x)$. We can then construct a truth table with columns $\mathbb{C} = atoms(P'_\sigma(x)) \cup \{P'_\sigma(x)\}$. For every $\alpha(x) \in \mathbb{C}$ and row r, we say $r \models \alpha(x)$ if and only if $\alpha(x)$ evaluates to $\top$ in row r of the table. For every row r, we define a conjunction consisting of all the information contained in row r as follows

$$\Psi_r(x) := \bigwedge_{\alpha(x)\in Atoms(P'_\sigma(x))} \begin{cases} \alpha(x) & \text{if } r \models \alpha(x) \\ \neg\alpha(x) & \text{otherwise} \end{cases}$$

We show that there is a CNF expression P^*_σ such that P'_σ is logically equivalent to P^*_σ.

Case 1. Consider first the case where $P_\sigma(x) \notin \mathbb{C}$. In other words, the equation $P'_\sigma(x)$ is not defined in terms of the input predicate $P_\sigma(x)$. Set $\psi_\sigma(x)$ and $\phi_\sigma(x)$ as follows.

$$\phi_\sigma(x) := \bigvee_{\{r : r \not\models P'_\sigma(x)\}} \Psi_r(x) \quad \text{and} \quad \psi_\sigma(x) := \bigvee_{\{r : r \models P'_\sigma(x)\}} \Psi_r(x) \tag{4.3}$$

Let $P^*_\sigma(x) =$ if $P_\sigma(x)$ then $\neg\phi_\sigma(x)$ else $\psi_\sigma(x)$. By construction, $\phi_\sigma(x) = \neg\psi_\sigma(x)$, and therefore $P^*_\sigma(x) \equiv \psi_\sigma(x)$. Because $\psi_\sigma(x)$ exhaustively contains all the possible situations in which $P'_\sigma(x)$ evaluates to $\top$, $\psi_\sigma(x) \equiv P'_\sigma(x)$. Thus, the CNF P^*_σ is equivalent to P'_σ.

Case 2. $P_\sigma(x) \in atoms(P'_\sigma(x))$. Define $\phi_\sigma(x)$ and $\psi_\sigma(x)$ as follows.

$$\phi_\sigma(x) := \bigvee_{\{r : r \models P_\sigma(x), r \not\models P'_\sigma(x)\}} \Psi_r(x) \quad \text{and} \quad \psi_\sigma(x) := \bigvee_{\{r : r \not\models P_\sigma(x), r \models P'_\sigma(x)\}} \Psi_r(x) \tag{4.4}$$

Set $P^*(x) =$ if $P_\sigma(x)$ then $\neg\phi_\sigma(x)$ else $\psi_\sigma(x)$. We show that $P^*_\sigma(x)$ evaluates to $\top$ iff $P'_\sigma(x)$ evaluates to $\top$. Consider first the case where $P_\sigma(x)$ evaluates to $\top$. Then $P^*_\sigma(x)$ evaluates to $\top$ iff $\phi_\sigma(x)$ evaluates to $\bot$. By the definition in (4.4), this means Ψ_r evaluates to $\bot$ for every row r such that $r \not\models P'_\sigma(x)$. In other words, none of the conditions under which $P'_\sigma(x)$ can evaluate to $\bot$ are satisfied. Thus, $\phi_\sigma(x)$ evaluates to $\bot$ iff $P'_\sigma(x)$ evaluates to $\top$. Similarly in the case where $P_\sigma(x)$ evaluates to $\bot$, $P^*_\sigma(x)$ evaluates to $\top$ iff $\psi_\sigma(x)$ evaluates to $\top$. By the definition in (4.4), $\psi_\sigma(x)$ evaluates to $\top$ iff $\Psi_r(x)$ evaluates to $\top$ for *some* row r. In other words, one of the conditions under which $P'_\sigma(x)$ can evaluate to $\top$ is satisfied. Thus, $\psi_\sigma(x)$ evaluates to $\top$ iff $P'_\sigma(x)$ evalautes to $\top$. □

4.1.2 Revisiting Phonological Generalizations

Section 3.3 showed how licensing and blocking structures have a natural BMRS representation. In the case of postnasal voicing, the grammatical surface forms are the result of a constraint *NC̥, which says a voiceless consonant cannot be immediately preceded by a nasal. This constraint was implemented in BMRS as a logical expression N**C̥** in (26b) and repeated below in (35). N**C̥**$(x) = \top$ iff x is voiceless consonant that is immediately preceded by a nasal. The output predicate [voi]$'(x)$ can be defined in such a way where N**C̥** is a *licensing* structure for the feature [voice]. This is repeated below in (35a).

The CNF proposed in this section deviates from representing output predicates with licensing and blocking structures. The normal form of the program which models postnasal voicing makes a concrete connection between the phonotactic constraint on surface forms, and the *process* that models the relationship between underlying and surface forms. More precisely: a voiceless consonant becomes voiced when it is preceded by a nasal as a means of *repairing* an ungrammatical sequence of sounds. When the word /pama/ 'clothing' in Zoque combines with the first person possessive morpheme /m-/, the resulting underlying form /mpama/ violates *NC̥. The surface form [mbama] in Figure 4.1 is the result of repairing the ungrammatical sequence of sounds by means of voicing the sound which leads to *NC̥ being violated. Thus, an equivalent way to express the generalization in (35a) is that N**C̥** expresses the *environment* which triggers voicing. This is precisely the generalization captured by the normal form of [voi]$'(x)$ in (35b). The key takeaway from (35) is that N**C̥** being a licensing structure for the feature [voice] is logically equivalent to N**C̥**

being the trigger environment for voicing. In this way, CNF programs encode the same phonological generalizations as programs written in terms of licensing and blocking structures.

(35) *Various ways to encode phonological generalizations using logic*

$\text{N}\mathring{\text{C}}(x) = \neg[\text{son}](x) \wedge \neg[\text{voi}](x) \wedge [\text{nas}](p(x))$

a. Licensing and Blocking: The $\text{N}\mathring{\text{C}}$ constraint is a licensing structure for the feature [voice]; if a voiceless consonant is preceded by a nasal, it will be voiced in the output.
$[\text{voi}]'(x) = \text{if } \text{N}\mathring{\text{C}}(x) \text{ then } \top \text{ else } [\text{voi}](x)$

b. Canonical Normal Form: The $\text{N}\mathring{\text{C}}$ constraint describes the environment in which voicing takes place; a voiceless consonant will become voiced if it is preceded by a nasal.
$[\text{voi}]'(x) = \text{if } [\text{voi}](x) \text{ then } \neg\bot \text{ else } \text{N}\mathring{\text{C}}(x)$

c. Disjunction: A sound is voiced in the output if it either violates the $\text{N}\mathring{\text{C}}$ constraint, or is voiced in the input.
$[\text{voi}]'(x) = \text{N}\mathring{\text{C}}(x) \vee [\text{voi}](x)$

One thing to note in the CNF program is the use of the `if...then...else` syntax. The equations in (35) shows that the use of this syntax is not necessary, since the equations in (35a,b) can more compactly be expressed with the disjunction in (35c). This is further shown with respect to the string function $a \rightarrow b/b\_\_$, where the output predicates $P'_a(x)$ and $P'_b(x)$ could both be expressed using conjunctions and disjunctions, as shown in (31) and (30), respectively. Moreover, CNF itself makes use of conjunctions and disjunctions in the 'then' and 'else' parts of an equation. This brings into question why we would bother using the `if...then...else` syntax at all. In the case of CNF programs, the syntax compartmentalizes each equation in a program into two components: the 'then' expression and the 'else' expression. The benefit of this for *learning* a BMRS representation of a string function is that the learning task is broken down into learning the 'then' and 'else' components for every feature. For any feature F, the 'then' ('else') part of the output predicate $F'(x)$ expresses the environment where a [+F] ([-F]) sound in the underlying form becomes [-F] ([+F]) in the surface form. In this way, learning programs which have the `if...then...else` syntax amounts to learning *phonological generalizations* regarding environments where features undergo change.

A further benefit of this syntactic form is further discussed in Chapter 5, where it is shown

that this syntactic structure makes it possible to break a logical transduction down into smaller transductions. More concretely, if a sample of underlying and surface form pairs are from two separate non-interacting processes, the learner will learn the *simultaneous application* of these two processes. There is a systematic way of parsing CNF programs into separate programs which model the separate processes that it is the simultaneous application of.

4.2 Learning Phonology

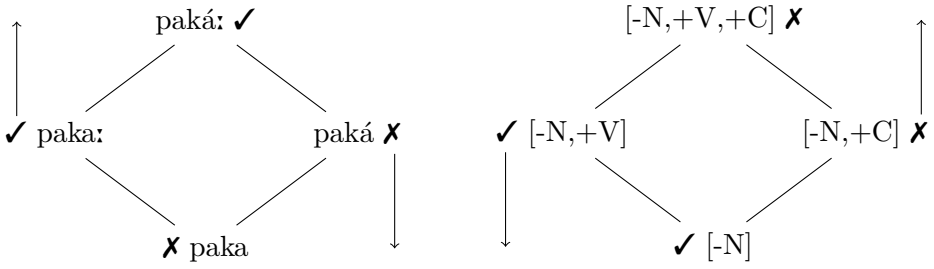
The normal form for ISL BMRS programs in Definition 4.1 formalizes the idea that in order to know how a function behaves, we must know exactly the changes that take place and the environments which trigger those changes. This is because the normal form associates each output predicate P'_σ with two pieces of information: the environment where a σ in the input becomes a non-σ in the output (ϕ_σ), and the environment where a non-σ in the input becomes a σ in the output (ψ_σ). For any index x, these two pieces of information are sufficient to determine whether $P'_\sigma(x)$ holds. This means that for an ISL function f over the alphabet Σ and predicates $\{P_\sigma\}_{\sigma\in\Sigma}$ the pieces of information $\{\phi_\sigma, \psi_\sigma\}_{\sigma\in\Sigma}$ uniquely identify f. The fact that every ISL function can be expressed by a BMRS program in normal form (Theorem 4.3) therefore means that learning the BMRS program which models f amounts to learning $\{\phi_\sigma, \psi_\sigma\}_{\sigma\in\Sigma}$. In Section 4.3 it is shown that each of these can be represented with a partially ordered hypothesis space which facilitates efficient learning. This section discusses the relevant concepts and background on phonological learning that motivates the learning procedure that will be presented in Section 4.3.

4.2.1 Partially-Ordered Hypothesis Spaces

Recall that a poset (previously defined as Definition 2.6) is a structure $(X, \leq)$ such that $\leq$ is a reflexive, anti-symmetric, and transitive relation over X. Posets have been used as a hypothesis space for various learning problems within phonology [Rawski, 2021, Chandlee et al., 2019, Tesar, 2013, Heinz et al., 2012, Heinz, 2010a]. A common theme among them is that a partially-ordered hypothesis space encodes entailment relations that are relevant for learning. These entailments allow a large space to be pruned efficiently with a small number of data points. The advantage of using a partially-ordered hypothesis space for learning is therefore efficiency. Two examples are

presented here to exemplify this point.

The problem of learning the underlying form of a particular surface form is an example of a learning problem that utilizes a partially-ordered hypothesis space. Tesar [2013] studies this problem within a stress-length system[1] where every vowel in a word can be stressed/unstressed and long/short. Consider the surface form [paká:], where the second vowel is both stressed and long. Within the stress-length system, there are 16 possible underlying forms for [paká:]. However, there is a natural ordering on the space of possible underlying forms based on relative similarity. The relative similarity relation on underlying forms is defined as follows: $x \leq y$ iff y is more similar to [paká:] than x. The space of all possible underlying forms of [paká:] with the relative ordering relation forms a poset. A small subset of the space is presented in Figure 4.4(a). The top element of the structure is the underlying form that is most similar to [paká] (i.e. the one in which no changes take place). The forms /paka:/ and /paká/ both differ from [paká:] by exactly one change. /paká/, for example, differs from the surface form with respect to the length of the final vowel. Thus, the ordering paká≤paká: in the poset represent the following relation: paká: is more similar to [paká:] than paká. Since neither /paká/ nor /paka:/ is more similar to [paká:] than the other, these two underlying forms are not ordered with respect to each other. Both of these forms are more similar to [paká:] than /paka/, and therefore both are above the form paka in the structure.



(a) Poset of possible underlying forms for [paká:], ordered by relative similarity [Tesar, 2013]. Optimal candidates (✓) are upward entailing; non-optimal candidates (✗) are downward entailing.

(b) Poset of feature specifications, ordered by generality [Rawski, 2021]. Grammatical structures (✓) are downward entailing; ungrammatical structures (✗) are upward entailing.

Figure 4.4: Examples of partially-ordered hypothesis spaces from Tesar [2013] and Rawski [2021], and the entailment patterns which facilitate efficient learning.

Although the poset contains 16 possible candidates for the underlying form of [paká:], there are entailment relations that efficiently shrink the possible space. Consider for example the candidate

/paká/. If the map /paká/→[paká:] leads to inconsistency, then we can rule out /paká/ a possible underlying form. However, this information also entails that anything below /paká/ in the relative similarity poset will lead to inconsistency, and can be ruled out as a possible underlying form. Thus, being inconsistent is downward entailing, as illustrated in Figure 4.4(a). The larger space containing all 16 candidates for [paká:] is presented in Figure 4.5. If the learner determines that /paká/ cannot be the underlying form, then the information gained from this is that the second vowel in [paká:] must be [+long] underlyingly, since that is the only distinction between the two forms. Thus the entire substructure highlighted in gray in Figure 4.5 is removed from the hypothesis space, pruning the entire space in half. Because of this, the space of possible underlying forms reduces from 16 candidates to just 8, even though inconsistency was only observed for a single candidate. As Tesar [2013] explains, the benefit of this is computational efficiency:

> Within a relative similarity lattice, then, the possible optimality of a node entails upward, while the definite non-optimality of a node entails downward. This makes it possible to draw conclusions about whole sublattices of candidates based on the evaluation of only a single candidate. [...] This converts exponential search into linear search: the number of possible underlying forms is exponential in the number of features, but the number of forms to actually be tested is linear in the number of features[...] [Tesar, 2013, pg.287-289]

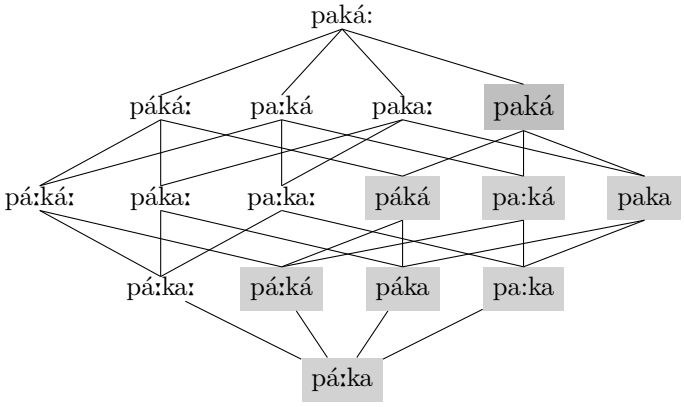


Figure 4.5: Full relative similarity lattice [Tesar, 2013, pg.189], where goal is to learn the underlying form of [paká:] If /paká/→[paká:] leads to inconsistency, then the final vowel cannot be [-long] in the underlying representation. The entire substructure below the candidate /paká/ (highlighted) is removed from the hypothesis space.

The substructure highlighted in Figure 4.5 is an *ideal* of the larger structure. The dual of an ideal is called an *filter*; these are presented as Definitions 4.4 and 4.5. The upward and downward entailment patterns which are relevant for learning allow the hypothesis space to be pruned very efficiently because with each new piece of information that is obtained through the learning process, an entire filter or ideal of the hypothesis space is removed. In Figure 4.5 the principle ideal (↓ paká) is removed from the hypothesis space.

[1]This work is specifically for output-driven maps, which are outside the scope of this dissertation.

Definition 4.4 (**Filters**)**.** For a poset $(X, \leq)$, a *filter* is a non-empty set $F \subseteq X$ such that:

(a) for every $x \in F$, if $x \leq y$ then $y \in F$ [upward closure]

(b) for every $x, y \in F$, there is some $z \in F$ such that $z \leq x$ and $z \leq y$ [downward directed]

For every $x \in X$, the **principle filter** generated by x (denoted $\uparrow x$) is the set of upper bounds of x.

$$\uparrow x := \{y \in X | x \leq y\}$$

Definition 4.5 (**Ideals**)**.** For a poset $(X, \leq)$, an *ideal* is a non-empty set $I \subseteq X$ such that:

(a) for every $x \in I$, if $y \leq x$ then $y \in I$ [downward closure]

(b) for every $x, y \in I$, there is some $z \in I$ such that $x \leq z$ and $y \leq z$ [upward directed]

For every $x \in X$, the **principle ideal** generated by x (denoted $\downarrow x$) is the set of upper bounds of x.

$$\downarrow x := \{y \in X | y \leq x\}$$

Interestingly, the problem of learning phonotactic constraints on what combinations of sounds are *not allowed* (ungrammatical) in a language can be formalized in exactly the same way. Chandlee et al. [2019] and Rawski [2021] study this problem over feature models. Consider the features [nasal], [voice], and [coronal], to be denoted with N, V, and C respectively. There is a natural ordering on the space of all feature specifications over these three features based on generality. A small subset of the space is presented in Figure 4.4(b). The top element [-N,+V,-C] is a maximally-specific feature specification in the sense that a binary value is specified for each feature. The specifications [-N,+V] and [-N,+C] are both more general than [-N,+V,-C]. The set of all sounds which satisfy [-N,+V], for example, form a superset of the sounds which satisfy [-N,+V,-C]. Thus the ordering [-N,+V]≤[-N,+V,-C] in the poset represents the following relation: the sounds with feature description [-N,+V,-C] are contained in the set of sounds with feature description [-N,+V]. With respect to this ordering, [-N,+V] and [-N,+C] are not ordered with respect to each other, but both are ordered above [-N] since the set of [-nasal] sounds is contained in the set of [-nasal,+voice] as well as [-nasal,+coronal] sounds.

Similar to the example of inferring underlying forms within a partial order, the poset in Figure 4.4(b) encodes entailment relations related to grammaticality. If a language does not allow [-nasal,+coronal] sounds, then it will also not allow [-nasal,+voice,+coronal] sounds. Thus, being ungrammatical is upward-entailing. On the other hand, if a feature specification is allowed in a

language, then everything below it in the structure will be allowed as well. Thus, grammaticality is downward-entailing. The full space of feature specifications over [nasal], [voice], and [coronal] is presented in Figure 4.6. If a language does not allow nasals, then [+N] and all the feature specifications *above* it in the structure are ungrammatical. If the goal is to learn constraints against *ungrammatical* structures, the entire principle filter ↑[+N], which is highlighted in gray in Figure 4.6, can be removed from the hypothesis space since they do not need to be considered. Rawski [2021] discusses this with respect to the role of filters and ideals:

> The structural filters give the learner an advantage when confronting hypothesis spaces under a particular model. In particular, it allows the learner to prune vast swathes of the hypothesis space as it reaches for principal elements of features. If a learner identifies one structure as being grammatical, the learner may infer that all of its factors are also grammatical and not have to consider them. Alternatively, if the learner knows a structure is ungrammatical, it may infer that the filters above it are also ungrammatical. [Rawski, 2021, pg. 48]

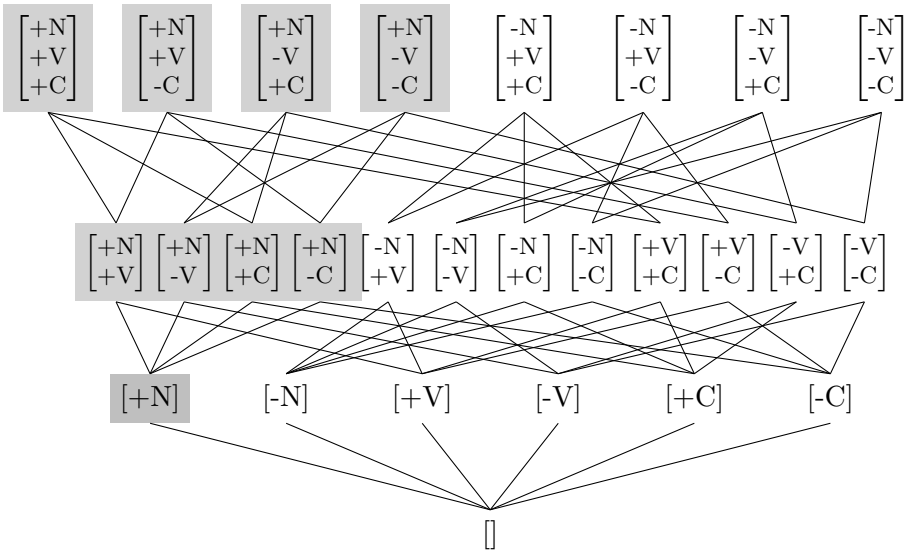


Figure 4.6: Full feature specification poset [Rawski, 2021, pg.44], where the goal is to learn the banned forms of language. If nasals are banned in a language, then all the sets of sounds above it in the structure (highlighted) must also be banned, and do not need to be considered. The structure above [+N] can therefore be removed from the hypothesis space.

The above discussions from Tesar [2013] and Rawski [2021] illustrate how two very different learning goals can be pursued with very efficient learning procedures by exploiting the structure induced by the relevant ordering on the set of possible candidates, whether they be underlying forms or feature specifications. The space of candidates that the learner must search through

have a partially-ordered structure which has significant computational advantages for the learning problem. The entailment relations encoded by ideals and filters within the hypothesis space allow the learner to remove large substructures with a single data point. This process of pruning the poset of feature specifications is analogous to that of pruning the space of underlying forms. In the former, ungrammaticality is upward-entailing and filters are removed from the space, while in the latter being inconsistent is downward-entailing and ideals are removed from the space. The guiding motivation for this chapter is that similar structures can be constructed for the problem of learning BMRS programs. Recall the postnasal voicing BMRS program from (33), where a [-voi] input becomes [+voi] if it is a [-son] that is preceded by a [+nas]. Since no other features undergo changes in this map, in order to learn the entire BMRS program which models postnasal voicing, it is sufficient to learn the environment which triggers voicing (i.e. $\psi_{[\text{voi}]}$). The CNF BMRS program that models postnasal voicing depends on this information, as highlighted in (36). Thus, the goal of learning the program which models postnasal voicing is to learn $\psi_{[\text{voi}]}(x)$.

(36) *Learning goal for postnasal voicing*

$$[\text{voi}]'(x) = \text{if } [\text{voi}](x) \text{ then } \neg\bot \text{ else } \psi_{[\text{voi}]}(x)$$
$$F'(x) \quad = \text{if } F(x) \quad \text{then } \neg\bot \text{ else } \bot \qquad (\equiv F(x)) \qquad \text{for all other features } F$$

The observation which connects this learning goal to the previous two examples is that the space of possible triggering environments for voicing can be represented by a poset. A small subset of possible candidates for $\psi_{[\text{voi}]}(x)$ are presented in Figure 4.7 to illustrate the analogy between this approach and the previous two discussed in this section. Because these are logical expressions, a formal way to represent the partial order between the candidates is *entailment*. In other words, the ordering $[\text{nas}](px) \leq \neg[\text{son}](x) \wedge [\text{nas}](px)$ in the poset represents the relation: $\neg[\text{son}](x) \wedge [\text{nas}](px)$ *entails* $[\text{nas}](px)$. Since neither $\neg[\text{voi}](x) \wedge [\text{nas}](px)$ nor $\neg[\text{son}](x) \wedge [\text{nas}](px)$ entails the other, these two are not ordered with respect to each other. And since $\neg[\text{son}](x) \wedge \neg[\text{voi}](x) \wedge [\text{nas}](px)$ entails both of these candidates, it is ordered above both.

A key observation in the posets discussed from Tesar [2013] and Rawski [2021] is that there are entailment relations within the structure that are useful for learning. A similar observation holds of the problem of learning environments where features undergo change. If an environment triggers voicing, then anything above it in the poset must also trigger voicing. That is, if being

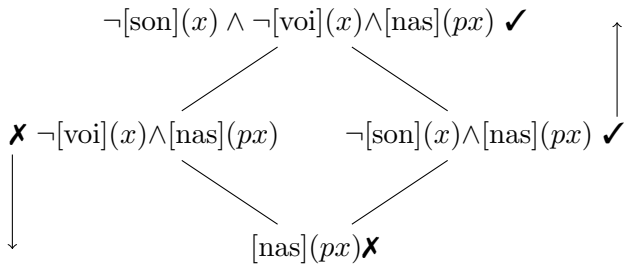


Figure 4.7: Poset of possible expressions for $\psi_{[\text{voi}]}(x)$ (environment where [-voice] input becomes [+voice] output). Triggering environments (✓) are upward entailing; non-triggering environments (✗) are downward-entailing.

an obstruent that is preceded by a nasal is *sufficient* to trigger voicing, then being a voiceless obstruent that is preceded by a nasal must also trigger voicing. In other words, being a triggering environment is *upward* entailing. Similarly, if an environment does *not* trigger voicing, then none of the environments below it will trigger voicing. In other words, if an environment is *no sufficient* to trigger voicing, then none of its subfactors are sufficient to trigger voicing. More specifically, if an environment is such that there is *some* example of an input-output which satisfies that environment and in which voicing does not take place, then *every* subfactor below it in the structure is also such that there is *some* input-output pair of words which satisfies it and in which voicing also does not take place. Thus, filters and ideals play a similar role in the structure in Figure 4.7 as they do in the two examples in Figure 4.4.

An equivalent way to think about the relation represented in the poset in Figure 4.7 is in terms of *generality*: [nas](px) is more general than ¬[son](x)∧[nas](px), which is more general than ¬[son](x) ∧ ¬[voi](x)∧[nas](px). Thus, the bottom of the poset represents the most general candidates for $\psi_{[\text{voi}]}(x)$, and the candidates become more specific (informative) as they go up the structure. This perspective on the space makes the analogy between the goal of learning triggering environments such as $\psi_{[\text{voi}]}(x)$ and learning a grammars in Chandlee et al. [2019]'s approach more concrete. This analogy is explored in more formal detail in the subsequent sections.

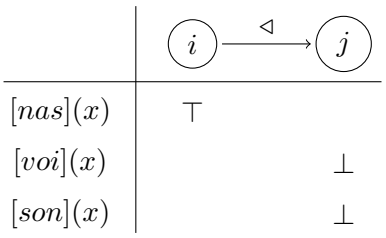
The example with features [nasal], [voice], and [coronal] in Figures 4.4(b) and 4.6 involve grammaticality of a single type of sound (e.g. whether a language permits voiced nasal coronals). The goal of the learning procedure presented by Chandlee et al. [2019] and Rawski [2021] is to learn constraints against *sequences* of sounds. Their learning procedure is discussed in more detail in Section 4.2.2 in terms of k-factors. Section 4.3 then shows how this work can be adapted for learning triggering environments and consequently, BMRS representations of phonological maps.

4.2.2 Bottom-Up Factor Inference Algorithm (BUFIA)

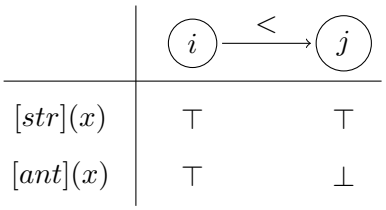
The learning algorithm introduced by Chandlee et al. [2019] is called the Bottom-Up Factor Inference Algorithm (BUFIA). The goal of BUFIA is to learn the *banned* k-factors (discussed in Section 2.1.2) of a language from a positive sample (i.e. collection of grammatical words from the language). Consider again the examples of postnasal voicing and sibilant harmony, which were discussed in Section 3.1.1. Postnasal voicing is a means of repairing violations of the OT constraint *NC̥, which says a nasal and a voiceless consonant cannot appear adjacent to each other [Pater, 1996, 1999]. *NC̥ is a description of *ungrammatical* surface forms. Within the model-theoretic framework discussed in Section 2.1, this constraint can be expressed in terms of banned *substructures*. In particular, because *NC̥ is a constraint on adjacent segments, it can be expressed as a ban on 2-factors in a successor model, where index i has feature [nasal] and index $i+1$ has features [-voi,-son]. Figure 4.8(a) illustrates the banned 2-factors.

Similarly for sibilant harmony, the phonotactic constraint which characterizes ungrammatical surface forms can be expressed as follows: [+ant] and [-ant] stridents cannot co-occur in a single word. In the framework of OT, this statement is formalized as a markedness constraint of the form *[+str,α ant]...[+str,-α ant] [Hansson, 2001, Rose and Walker, 2004]. The underlying form /skutiwaʃ/ 'he saw' is ungrammatical in Inseño Chumash because the initial sibilant /a/ is [+ant] while the second sibilant /ʃ/ is [-ant]. Harmony takes place in order to repair this constraint violation and only the surface form [ʃkutiwaʃ] is observed. In the model-theoretical framework, this constraint is expressed as a ban on 2-factors in a precedence model, where an index i with features [+str,$\pm$ant] precedes an index j with features [+str,$\mp$ant]. Figure 4.8(b) illustrates the banned 2-factors expressed by this constraint.

Recall the discussion of upward and downward entailment presented in Figure 4.4(b); if a feature specification is ungrammatical in a language, then everything above it in the poset is ungrammatical. This concept extends to k-factors; if some k-factor is ungrammatical, then all the structures above it in the poset (i.e. its superfactors) are also ungrammatical. BUFIA runs as follows. There is a finite sample S containing words from a language. The goal is to learn the *grammar* of the language, which corresponds with the *phonotactic constraints* of the language. In other words, the goal is to learn the k-factors which are not permitted in the language. Moreover,



(a) The 2-factor that identifies words which violate the OT constraint *NÇ.



(b) The 2-factor that identifies words which violate the constraint *[+str,+ant]...[+str,-ant].

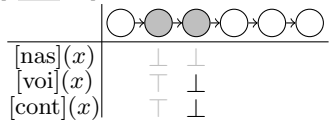
Figure 4.8: Examples of 2-factors in feature models that identify ungrammatical words that violate surface form constraints. The *NÇ constraint involves two adjacent sound; it correspond with 2-factors in a successor model. The *[+str,+ant]...[+str,-ant] constraint involves two sounds that can have any number of sounds intervene; it corresponds with 2-factors in a precedence model.

because there are many ways to express the ungrammatical k-factors, a further goal is to find the most general description. For this reason, the algorithm searches through the hypothesis space *bottom-up*. If some k-factor X in the space is not found in any surface form in S, then it is added to the set of constraints (i.e. grammar) of the language. Because ungrammaticality is upward entailing, all of X's superfactors must also be ungrammatical. Thus, the entire principle filter $\uparrow X$ is removed from the hypothesis space (similar to Figure 4.6) since the grammaticality of these structures no longer needs to be considered. In this way, only the minimal (most general) ungrammatical k-factors are part of the resulting grammar.

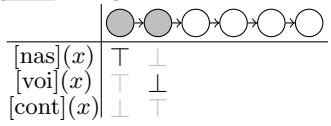
BUFIA is briefly illustrated here with the example of postnasal voicing in Zoque, originally presented in Section 3.1.1. Some examples of grammatical surface forms in Zoque are presented in (37). To keep the discussion short, we are only considering a small subset of 2-factors over the domain of features [nasal], [voice], and [continuant], given in Figure 4.9. For brevity, the feature [sonorant] is left out of these models because it is ultimately not necessary to identify the minimal ungrammatical 2-factor in Zoque.

(37) *Some models of grammatical surface forms in Zoque* [Wonderly, 1951]

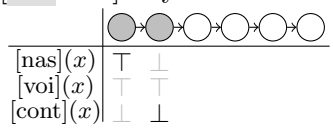
a. [t at ah] 'father'



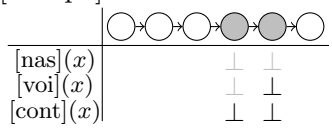
b. [nh ayah] 'my husband'



c. [nd atah] ‘my father’



d. [ku?tpa] ‘he eats’



Recall the transformation /mpama/→[mbama] from Figure 4.1. The surface form [mpama] is ungrammatical in Zoque because of the adjacent /mp/ sounds. With respect to the features [nas], [voi], and [cont], /mp/ corresponds with the 2-factor [+nas,+voi,-cont][-nas,-voi,-cont]. This 2-factor is the top element of the poset presented in Figure 4.9. The surface forms in (37a-c) provide evidence that [+nas][-voi,-cont] is the smallest subfactor of /mp/ which characterizes the ungrammatical surface forms in Zoque. Consider first the surface form [tatah] ‘father’ in (37a); the /at/ sequence of sounds is evidence that the 2-factor [-nas,+voi,+cont][-nas,-voi,-cont] is grammatical in Zoque. Consequently, all its subfactors are grammatical as well. In particular, the 2-factor [][-voi,-cont] in Figure 4.9 is grammatical. Similarly, the /nh/ sequence of sounds in [nhaya] ‘my husband’ in (37b) provide evidence that the 2-factor [+nas,+voi,-cont][-nas,-voi,+cont] and all its subfactors are grammatical. Thus the 2-factor [+nas][-voi] in Figure 4.9 is grammatical. The /nd/ sequence in [ndatah] ‘my father’ in (37c) provides evidence that the 2-factor [+nas,+voi,-cont][-nas,+voi,-cont] and all its subfactors are grammatical. In particular, the 2-factor [+nas][-cont] in Figure 4.9 is grammatical. The final form in (37d) will be relevant for learning the postnasal voicing function in Section 4.3. Thus, the three surface forms in (37a-c) provide evidence that all the subfactors below [+nas][-voi,-cont] in Figure 4.9 are grammatical. This means that [+nas][-voi,-cont] is the *minimal* structure in the poset that is not found in any surface form of Zoque. All the structures below it are grammatical, while all the structures above it ungrammatical. For example, the 2-factors [+nas][-nas,-voi,-cont] and [+nas,+voi][-voi,-cont] are both also valid descriptions of ungrammatical 2-factors in Zoque. Both of these are, however, entailed by [+nas][-voi,-cont]. A similar poset for the subfactors and superfactors of the the ungrammatical Chumash 2-factor in Figure 4.8(b) can be found in Chandlee et al. [2019].

The structures in Figure 4.9 are notably missing the feature [sonorant]. However, adding this feature to the hypothesis space would not make a difference. Consider again the surface form [nhaya] in (37b). The [-voi,-son] /h/ does not have a voiced counterpart in Zoque, and therefore

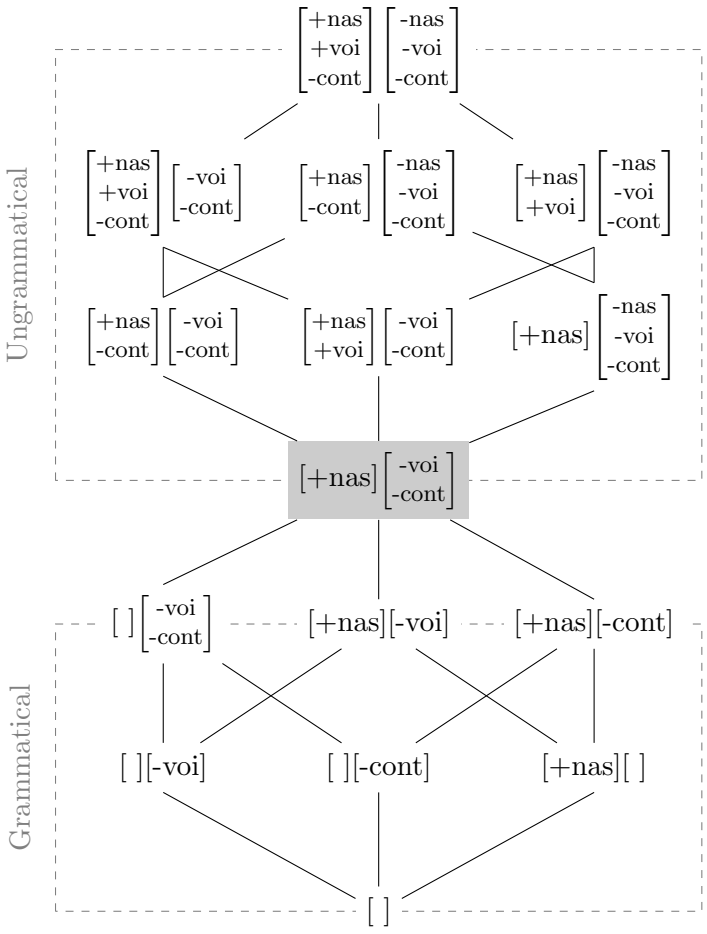


Figure 4.9: Poset of 2-factors above and below the structure in Figure 4.8(a) (highlighted), which is the minimal element in the poset that describes the ungrammatical surface forms in Zoque.

does not become voiced after a nasal [Wonderly, 1951]. Thus, [nhaya] provides evidence that [+nas][-voi,-son] (and every subfactor of it) is grammatical. The 2-factor [+nas][-voi,-son] therefore does not suffice as a constraint in Zoque, and the feature [continuant] is necessary to distinguish the sounds which can be preceded by a nasal from the sounds which cannot. Moreover, since the ungrammaticality of [+nas][-voi,-cont] entails the ungrammaticality of [+nas][-voi,-son,-cont], the [sonorant] feature is not necessary to represent the ungrammatical structures in Zoque.

The Zoque data points in (37) are revisited in Section 4.3 to show how the same structures and principles can be used to learn the postnasal voicing *map* in Zoque. Further discussion of BUFIA and abduction principles used in learning can be found in Rawski [2021]. Recent applications of BUFIA can also be found in Li [2025] and Payne [2024].

4.3 Adapting Phonotactic Learning to Transformations

This section presents an approach to this learning problem for *length-preserving ISL functions over feature models*. We further assume that there is one-to-one correspondence between indices of the input model and the output model; for every index x in the input model, its corresponding output is index x of the output model. The main idea expressed here is that for a set of features $\mathbb{F}$ and each of the pieces of information in $\{\phi_F, \psi_F\}_{F\in\mathbb{F}}$, there is a separate hypothesis space that can be represented as a *partial order*. Each of these expressions is learned through a process of pruning the corresponding hypothesis space. Because knowing $\{\phi_F, \psi_F\}_{F\in\mathbb{F}}$ is sufficient for knowing the BMRS program that models a function (Theorem 3.6), this means that a BMRS program can be learned using poset hypothesis spaces.

Recall that the positive sample for BUFIA is a set of grammatical surface forms, which are represented as models. In the case of learning *maps*, the sample is instead a set of input-output pairs, which are represented as *pairs* of models. The goal of learning is to then determine the program which yields the relationship between the input and output structures. This section discusses the jump from learning phonotactics to learning transformations, and how these two are inherently related. Section 4.3.1 goes through the algorithm with a discussion of the hypothesis space and update functions, and Section 4.3.2 gives a case study with the postnasal voicing data in Zoque that was discussed in Section 4.2.2.

4.3.1 Hypothesis Space and Updates

Representation of k-Factors

The k-factors in BUFIA represent surface forms because the goal of learning is to infer the ungrammatical forms. When the goal of learning is instead to infer *maps*, the k-factors encode *environments* where change takes place. Consider again the *NC̥ constraint which bans substructures [+nas][-voi,-son] (previously presented in Figure 4.8a). Section 3.3 briefly discussed the typology of *NC̥ effects, in which languages have a variety of ways of repairing *NC̥ violations. Postnasal voicing, for example, targets the second sound in the NC̥ substructure; if a voiceless consonant is preceded by a nasal, it becomes [+voice] as a means of repairing the *NC̥ violation. Denasalization, on the other hand, targets the first sound in the NC̥ substructure; if a nasal is followed by a voiceless

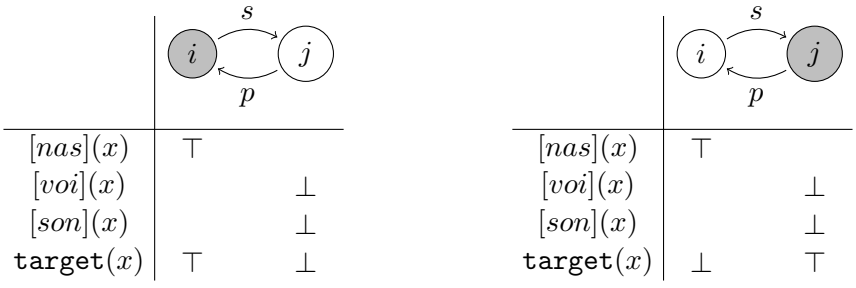
consonant, it becomes [-nasal] in order to repair *NC̥ violation. This is precisely why Chandlee and Jardine [2021] represent *NC̥ in BMRS with two separate expressions, repeated below. (26a) represents an instance of *NC̥ repair where the first sound is targeted (e.g. denasalization), and (26b) represents an instance where the second is targeted (e.g. postnasal voicing).

(26) *The *NC̥ constraint as BMRS expressions* [Chandlee and Jardine, 2021]

a. $\mathbf{N}\underset{\circ}{\mathbf{C}}(x) = \text{if } [nas](x) \text{ then } \big(\neg[son](s(x)) \wedge \neg[voi](s(x))\big) \text{ else } \bot$

b. $\mathrm{N}\underset{\circ}{\mathbf{C}}(x) = \text{if } \big(\neg[son](x) \wedge \neg[voi](x)\big) \text{ then } [nas](p(x)) \text{ else } \bot$

With respect to k-factors, the difference between postnasal voicing and denasalization can be expressed as follows: denasalization targets the first index in the [+nas][-voi,-son] 2-factor, while postnasal voicing targets the second index. Thus, while the [+nas][-voi,-son] 2-factor is a sufficient description of an ungrammatical substructure, we need a way to encode the *targeted* index in order to have a description of phonological maps. Every k-factor we consider will therefore have an extra property that distinguishes it as the target. Figure 4.10 presents the 2-factor that represents *NC̥ with an additional predicate `target` which picks out the index that a phonological map targets. The 2-factor which represents the environment where denasalization takes place is (a), while the 2-factor which represents where postnasal voicing takes place is (b).



(a) First index is the target of repair. (b) Second index is the target of repair.

Figure 4.10: 2-factor representation of the *NC̥ constraint, with an additional predicate that encodes which index is the target of a phonological map that repairs *NC̥ violations.

As a shorthand notation, we represent the two factor in Figure 4.10(a) as [+nas][-voi,-son], and the 2-factor in (b) as [+nas][-voi,-son]. Each of these corresponds with a different logical expressions, summarized in Figure 4.11. The logical expression which describes the [+nas][-voi,-son] substructure is precisely the $\mathbf{N}\underset{\circ}{\mathbf{C}}(x)$ expression (26a). Similarly the logical expression which

describes the [+nas][-voi,-son] structure is the $\mathrm{N}\underset{\circ}{\mathbf{C}}(x)$ expression in (26b).

Process	Learning	2-factor	Logical expression
[+nas] → [−nas] /_[-voi,-son]	$\phi_{[nas]}$	[+nas][-voi,-son]	$[nas](x) \wedge \neg[voi](sx) \wedge \neg[son](sx) \equiv \mathbf{N}\underset{\circ}{\mathbf{C}}(x)$
[-voi,-son] → [+voi] / [+nas]_	$\psi_{[voi]}$	[+nas][-voi,-son]	$[\mathrm{nas}](px) \wedge \neg[\mathrm{voi}](x) \wedge \neg[\mathrm{son}](x) \equiv \mathrm{N}\underset{\circ}{\mathbf{C}}(x)$

Figure 4.11: Comparison of denasalization and postnasal voicing maps in terms of learning goals, targeted 2-factors, and the logical formula which expresses the environment where the relevant change takes place.

Recall the poset in Figure 4.7 which contained logical formulas that expressed possible environments which trigger postnasal voicing. For example, the element $[\mathrm{nas}](px) \wedge \neg[\mathrm{voi}](x) \wedge \neg[son](x)$ represents a voiceless obstruent which is preceded by a nasal. We can express this as the 2-factor [+nas][-voi,-son], where the underline indicates the segment which is the target of the map. The poset in 4.7 can therefore be rewritten in terms of 2-factors, represented in Figure 4.12. The two figures capture exactly the same idea; being a triggering environment for a [-voi]→[+voi] change is upward entailing, while *not* being a triggering environment is downward entailing. In general, every 2-factors can be converted to a logical formula, and vice versa. The formal details of how this translation is defined are left out of the discussion, as the idea is intuitively clear.

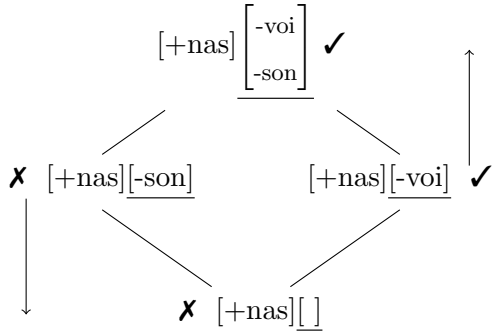


Figure 4.12: Poset of possible environments for $\psi_{[\mathrm{voi}]}$, represented as 2-factors. The underlined factor represents the target of voicing. Triggering environments (✓) are upward entailing; non-triggering environments (✗) are downward-entailing. This figure is equivalent to Figure 4.7.

It is important to note that the `target` predicate also ensures that the subfactor relation in the poset represents a *subenvironment* relation that is relevant for the learning goal. For example, [+nas][-voi] is a subfactor of [+nas][-voi,-son], but it is *not* a subfactor of [+nas][-voi,-son]. On the other hand, [+nas][-voi,-son] is a subfactor of [+nas][-voi,-son], even though it does not encode an environment. This is discussed further in the next section on hypothesis spaces.

Hypothesis Space

Since BUFIA learns ungrammatical surface forms, there is a single hypothesis space containing all the k-factors which are candidates for the grammar. In the case of learning transformations, there are several hypothesis spaces: one for each of the learning targets $\{\phi_F, \psi_F\}_{F\in\mathbb{F}}$. Moreover, for every $X \in \{\phi_F, \psi_F\}_{F\in\mathbb{F}}$ we need two sets of k-factors: $V(X)$ and $\hat{V}(X)$. $V(X)$ contains the relevant k-factors in which the targeted change takes place. Consider for example ϕ_F for some feature F. $V(\phi_F)$ contains all the k-factors in which an index x is such that $F(x) = \top$ in the input model but $F'(x) = \bot$ in the output model. $\hat{V}(\phi_F)$ contains all the k-factors in which an index x is such that $F(x) = \top$ in the input model and $F'(x) = \top$ in the output model. In other words, $V(\phi_F)$ contains all the k-factors in which a [+F] sound becomes [-F], and $\hat{V}(\phi_F)$ contains all the k-factors in which [+F] sounds do not undergo change. In this way, the collections V and $\hat{V}$ encode positive and negative evidence for the environments where features undergo change. The hypothesis spaces are then defined as in Definition 4.6.

Definition 4.6. For every $X \in \{\phi_F, \psi_F\}_{F\in\mathbb{F}}$, the corresponding **hypothesis space** is the set

$$\mathcal{H}(X) := \bigcup_{x\in V(X)} \downarrow x - \bigcup_{x\in\hat{V}(X)} \downarrow x \tag{4.5}$$

subject to the following two restrictions:

(i) Every $\mathcal{M} \in \mathcal{H}(X)$ has exactly one index i in the domain of $\mathcal{M}$ such that $\texttt{target}(i) = \top$.

(ii) If X is ϕ_F, then for every $\mathcal{M} \in \mathcal{H}(X)$, $F(i) = \top$ for the unique i such that $\texttt{target}(i) = \top$. If X is ψ_F then $F(i) = \bot$ for the unique i such that $\texttt{target}(i) = \top$

The posets defined in Definitions 4.6 takes all the k-factors in which the relevant change was observed in some pair of input-output models, and removes the subfactors in which the relevant change was *not* observed. The restrictions in (i) and (ii) then ensure that only the subfactors which represent relevant environments are included in the space. The restriction in (i) ensures that every structure in the hypothesis space represents an *environment*. Consider again the 2-factor [+nas][-voi]. The second index of this 2-factor is the unique index for which `target` holds. The 2-factor [+nas][-voi] which does not have any index as the target is a subfactor of [+nas][-voi]. However, [+nas][-voi] does not describe an environment and cannot be converted to a logical ex-

pression that represents an environment. Thus, condition (i) ensures that the space only includes the substructures which represents environments by ensuring that every model in the space has a unique index which is the target. The restriction in (ii) ensures that the hypothesis space includes only the *relevant* environments. If the goal is to learn the environment where a [-voi] sound becomes [+voi], then the hypothesis space should only include environments where the target of the change is [-voi]. This means that $\mathcal{H}(\psi_{[voi]})$ will only include k-factors where the unique index that satisfies the predicate `target` is such that $[voi](x) = \bot$.

Although the hypothesis space can be very large depending the the fixed value k and the number of features in $\mathbb{F}$, we only ever need to keep track of the minimal elements of the hypothesis space. The minimal elements correspond with the most general hypothesis that accounts for all the input-output pairs seen so far. Thus, for every $X \in \{\phi_F, \psi_F\}_{F \in \mathbb{F}}$, we have a collection of models $M(X) := min(\mathcal{H}(X))$. For each pair of input-output models, the relevant sets of structures V, $\hat{V}$, and M are updated. The update processes is discussed below.

Updating the Hypothesis

The learning procedure starts with an initial hypothesis, and generates a new hypothesis with each input-output pair of models. Before any data have been observed, $V(X)$ and $\hat{V}(X)$ are empty for each $X \in \{\phi_F, \psi_F\}$. This means that the initial hypothesis space for each X will also be empty by definition, and therefore the set of minimal elements $M(X)$ will be empty. In this case, the corresponding logical formula for X must be $\bot$. For example, if no instance of a [-voi]→[+voi] change is observed in the sample of input-output pairs, then $min(\mathcal{H}(\psi_{[voi]})) = \emptyset$. The hypothesis that corresponds with this is that [-voi] never comes [+voi]; in other words, $\psi_{[voi]}(x) = \bot$. The initial variables, hypothesis, and corresponding BMRS program are summarized in (38). The consequence of V and $\hat{V}$ being empty is that the initial (null) hypothesis corresponds with the BMRS program in (38c). This program is logically equivalent to the identity map. Thus, the initial hypothesis is that no features undergo change.

(38) *Initial Variables and Hypothesis*

a. *Variables;* for all $X \in \{\phi_F, \psi_F\}_{F \in \mathbb{F}}$

$$\begin{aligned} V(X) &= \emptyset \\ \hat{V}(X) &= \emptyset \\ M(X) &= \emptyset \end{aligned}$$

b. *Individual learning goals*
$\{\phi_F(x) = \bot;\ \psi_F(x) = \bot\}_{F\in\mathbb{F}}$

c. *Initial Hypothesis as a BMRS program*
$\{F'(x) = \text{if } F(x) \text{ then } \neg\bot \text{ else } \bot\}_{F\in\mathbb{F}}$
$\equiv \{F'(x) = F(x)\}_{F\in\mathbb{F}}$

When an input-output pair provides evidence for a feature change, the corresponding hypothesis space is updated. For example, if a pair of models $(\mathcal{M}, \mathcal{M}')$ is such that some index x is [-F] in the input model but [+F] in the output model, the set $V(\psi_F)$ will be updated with the k-factor(s) of $\mathcal{M}$ which contain x as the target. If, on the other hand, some index x is [-F] in the input and remains [-F] in the output, the set $\hat{V}(\psi_F)$ will be updated with the relevant k-factors. Each time V or $\hat{V}$ is updated, the set of minimal elements must also be updated. Thus, there are two possible types of updates.

Consider first the case in which a k-factor x is added to $V(X)$. In this case, the hypothesis space $\mathcal{H}(X)$ grows, and potentially the set of minimal elements grow. This means that when a new k-factor x is added to $V(X)$, the update function must maintain all previous minimal elements and only determines whether to add any new ones. The procedure for determining which elements is as follows: we traverse the structure $\downarrow x$ bottom-up in order to find the minimal subfactors of x which are neither subfactors nor superfactors of any current minimal elements. Because both k and the number of features we consider are fixed, the structure $\downarrow x$ is always of fixed size. Thus, determining the new set of minimal elements of the updated hypothesis space always involves a bottom-up search of a space of fixed size, even when the hypothesis space grows.

In the case where a new k-factor x is added to $\hat{V}(X)$, the update function must remove any element $m \in M(X)$ such that $m \sqsubseteq x$, and potentially add new minimal elements. In this case, the next minimal element will always be one level up in the poset. In other words, if m is no longer a minimal element, the new minimal element(s) will be immediately above m in the structure. This fact is illustrated in the case study in Section 4.3.2. Since the number of features is fixed, the number of possible structures immediately above m is fixed. Thus, similar to the previous case, updating the set of minimal elements involves traversing a space of fixed size.

The next section runs through the example of postnasal voicing in Zoque to illustrate how the hypothesis space, minimal elements, and hypothesis get updated incrementally after each input-output pair of models.

4.3.2 Case Study: Postnasal Voicing in Zoque

This section runs through the learning procedure using the example of postnasal voicing in Zoque, which was original presented as (14) in Section 3.1.1. A larger set of of data points from Zoque is presented in (39). The pairs of corresponding words in (a) and (b) show that voiceless sounds /p,t,k/ undergo voicing when they are preceded by a nasal, but /h,y,w/ do not. In previous sections, postnasal voicing was discussed as a map that targets [-voi,-son] sounds. In the case of Zoque, these examples show that it is the [-voi,-cont] sounds that undergo voicing in Zoque. This section shows how this generalization about Zoque can be inferred using the learning procedure in the previous section. This section moreover illustrates how the hypothesis space for learning the environment where voicing takes place is analogous to the hypothesis space for learning the ungrammatical surface forms. In Section 4.2.2, it was shown that the 2-factor [+nas][-voi,-cont] is the most general description of the *ungrammatical surface forms in Zoque*. This section shows that [+nas][-voi,-cont] is the most general description of the *environment where voicing takes place in Zoque*.

(39) *Postnasal Voicing in Zoque [Wonderly, 1951]*

a. pama 'clothing'	b. m**b**ama 'my clothing'
tatah 'father'	n**d**atah 'my father'
kama '(corn)field'	ŋ**g**amá 'my (corn)field'
hayah 'husband'	n**h**aya 'my husband'
yomo 'wife'	n**y**omó 'my wife'
wakas 'cow'	n**w**akás 'my cow'

We are going to assume that we are learning an ISL-2 function, and only consider 2-factors in our hypothesis space. Consider the underlying and surface form pair (/m**p**ama/, [m**b**ama]); the input-output *models* are presented in Figure 4.13. The sound at index 1 is [-voi] in the input model and [+voi] in the output model. This means that the set $V(\psi_{[voi]})$ must be updated with the 2-factors that contain the sound at index 1 as a target. There are two possible 2-factors in the input model which contain the sound at index 1. These are presented in Figure 4.14. The updated hypothesis space is presented in Figure 4.15. The corresponding updates are summarized in (40).

Since being [-voi] is presumed to be true when learning $\psi_{[voi]}$, the minimal element [-V] corresponds with the hypothesis $\psi_{[voi]}(x) = \top$. We could also set $\psi_{[voi]} = \neg[voi](x)$; this would give a logically equivalent program and corresponding map, but with unnecessary redundancy. The new

	m	p	a	m	a
	0	1	2	3	4
[nas](x)	$\top$	$\bot$	$\bot$		
[voi](x)	$\top$	$\bot$	$\top$		
[cont](x)	$\bot$	$\bot$	$\top$		
[nas]$'(x)$	$\top$	$\bot$	$\bot$		
[voi]$'(x)$	$\top$	$\top$	$\top$		
[cont]$'(x)$	$\bot$	$\bot$	$\top$		
	m	b	a	m	a

Figure 4.13: Input-output models for the map /mpama/→[mbama]; the sound at index 1 undergoes voicing. Some subfactor of the input string /mpama/ must be the environment that triggers voicing.

	m	p
[nas](x)	$\top$	$\bot$
[voi](x)	$\top$	$\bot$
[cont](x)	$\bot$	$\bot$
`target`(x)	$\bot$	$\top$

	p	a
[nas](x)	$\bot$	$\bot$
[voi](x)	$\bot$	$\top$
[cont](x)	$\bot$	$\top$
`target`(x)	$\top$	$\bot$

Figure 4.14: The 2-factors which get added to the set $V(\psi_{[voi]})$ given the input-output pair (/m**p**ama/, [m**b**ama]) in Figure 4.13.

hypothesis therefore says that a [-voi] sound *always* becomes [+voi]. This hypothesis corresponds with the *most general* hypothesis that accounts for the input-output sample observed.

(40) *Hypothesis updates given /mpama/→[mbama]*

$\mathcal{H}(\psi_{[voi]})$	Figure 4.15	
$min(\mathcal{H}(\psi_{[voi]}))$	$\{\underline{[-V]}\}$	
$\psi_{[voi]}(x)$	$\top$	
	$[nas]'(x)$	= if $[nas](x)$ then $\neg\bot$ else $\bot$
BMRS program	$[voi]'(x)$	= if $[voi](x)$ then $\neg\bot$ else $\top$
	$[cont]'(x)$	= if $[cont](x)$ then $\neg\bot$ else $\bot$
Phonological map	$[-voi] \rightarrow [+voi]$	

The previous update showed what happens when a feature change is observed. We consider next the case where the underlying and surface forms are the same. Consider the surface form [tatah] 'father'. The hypothesis in (40) predicts that we should not see this surface form in Zoque; we would instead expect to see [dadah]. This surface form therefore provides evidence that the the previous hypothesis is incorrect/too general, and needs to be updated. In particular, the second /t/ sounds indicates that neither being preceded nor followed by a [-N,+V,+C] sound is sufficient to trigger voicing. Thus, ↓[-N,+V,+C]$\underline{\text{[-N,-V,-C]}}$ and ↓$\underline{\text{[-N,-V,-C]}}$[-N,+V,+C] must both be removed from the hypothesis space. The updated space is presented in Figure 4.16. The updated hypothesis is presented in (41). Given both /mpama/→[mbama] and /tatah/→[tatah], the most

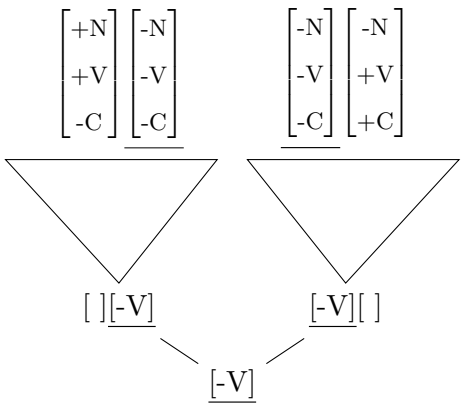


Figure 4.15: Updated hypothesis space $\mathcal{H}(\psi_{[voi]})$ given the two 2-factors added to $V(\psi_{[voi]})$ in Figure **??**, and the restrictions (i) and (ii) in Definition 4.6.

general hypothesis that accounts for *both* of these pairs of underlying and surface forms is that a [-voi] sound becomes voiced when it is preceded by either a [+nas] or a [+cont] sound.

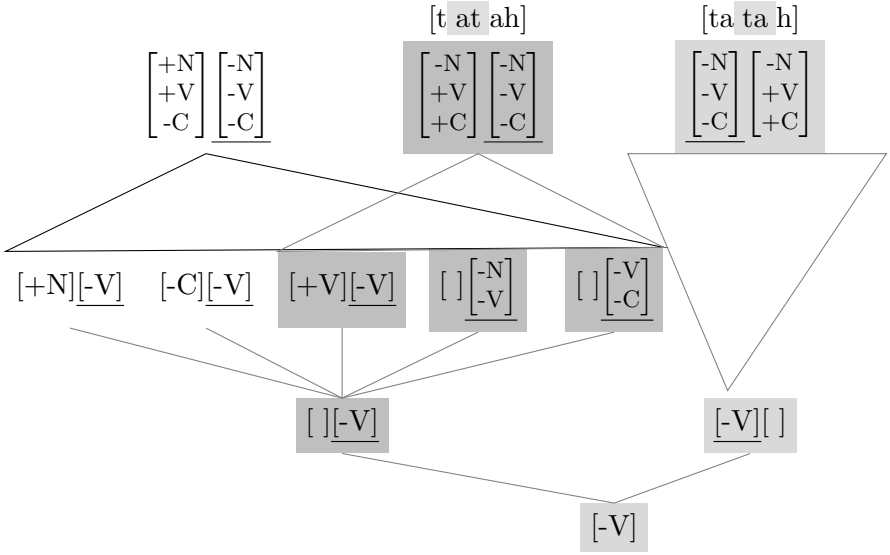


Figure 4.16: Updated hypothesis space after observing [tatah]. Principle filters ↓[-N,+V,+C][-N,-V,-C] and ↓[-N,-V,-C][-N,+V,+C] are removed from $\mathcal{H}(\psi_{[voi]})$.

(41) *Hypothesis updates given /tatah/→[tatah]*

$\mathcal{H}(\psi_{[voi]})$	Figure 4.16	
$min(\mathcal{H}(\psi_{[voi]}))$	$\{[+N][-V], [-C][-V]\}$	
$\psi_{[voi]}(x)$	$[nas](px) \vee [cont](px)$	
BMRS program	$[nas]'(x)$	= if $[nas](x)$ then $\neg\bot$ else $\bot$
	$[voi]'(x)$	= if $[voi](x)$ then $\neg\bot$ else $[nas](px) \vee [cont](px)$
	$[cont]'(x)$	= if $[cont](x)$ then $\neg\bot$ else $\bot$
Phonological map	$[-voi] \rightarrow [+voi] / \left\{ \begin{matrix} [+nas] \\ [-cont] \end{matrix} \right\} —$	

Consider next the surface form [nhayah] 'my husband'. Here we observe that the [-voi] sound /h/ also does not undergo voicing. This is because the /h/ sound in Zoque does not have a voiced counterpart [Wonderly, 1951]. Thus, the current hypothesis is incorrect because /h/ is preceded by a sound that is both [+nas] and [-cont], but it does not undergo voicing. Thus we remove the principle filter ↓[+N,+V,-C]<u>[-N,-V,+C]</u> from the hypothesis space since the [n h] combination in [nhayah] provides evidence that this 2-factor is not an environment that triggers voicing. The updated hypothesis space is presented in Figure 4.17. The updated hypothesis is presented in (42). Given the three input-output pairs discussed so far, the most general hypothesis that accounts for them is that voiceless [-cont] sounds undergo change if they are preceded by either a [+nas] or a [-cont] sound. Intuitively, this new hypothesis makes sense because the [nh] combination provides evidence that the voicing process does not apply to [+cont] sounds. The BMRS program in (42) moreover shows why it makes sense for expressions ψ_F and ϕ_F to be in disjunctive normal form. Each subfactor in the hypothesis space translates to a *conjunction*. If there is more than one minimal element, then we take the disjunction of each of the individual conjunctions. Because $\mathcal{H}(\psi_{[voi]})$ in Figure 4.17 has two minimal elements, the resulting expression for $\psi_{[voi]}(x)$ is a disjunction of two conjunctions.

(42) *Hypothesis updates given /nhayah/→[nhayah]*

$\mathcal{H}(\psi_{[voi]})$	Figure 4.17
$min(\mathcal{H}(\psi_{[voi]}))$	$\{[+N]\underline{[-V,-C]}, [-C]\underline{[-V,-C]}\}$
$\psi_{[voi]}(x)$	$([nas](px) \wedge \neg[cont](x)) \vee ([cont](px) \wedge \neg[-cont](x))$
BMRS program	$[nas]'(x) = \text{if } [nas](x) \text{ then } \neg\bot \text{ else } \bot$ $[voi]'(x) = \text{if } [voi](x) \text{ then } \neg\bot$ $\text{else } ([nas](px) \wedge \neg[cont](x)) \vee ([cont](px) \wedge \neg[cont](x))$ $[cont]'(x) = \text{if } [cont](x) \text{ then } \neg\bot \text{ else } \bot$
Phonological map	$[-voi,-cont] \rightarrow [+voi] / \left\{ \begin{matrix} [+nas] \\ [-cont] \end{matrix} \right\} \_\_$

The final surface form we consider is [ku?ta] 'he eats', which was previously presented in (37). The [?t] combination in this word provides evidence that being preceded by a [-cont] sound is not sufficient to trigger voicing; otherwise we would expect the surface form [?d]. Thus, we update the hypothesis space by removing the principle filter ↓[-N,-V,-C]<u>[-N,-V,-C]</u>, which represent the 2-factor for [?d]. The resulting hypothesis space is presented in Figure 4.18, and the resulting updates are presented in (43).

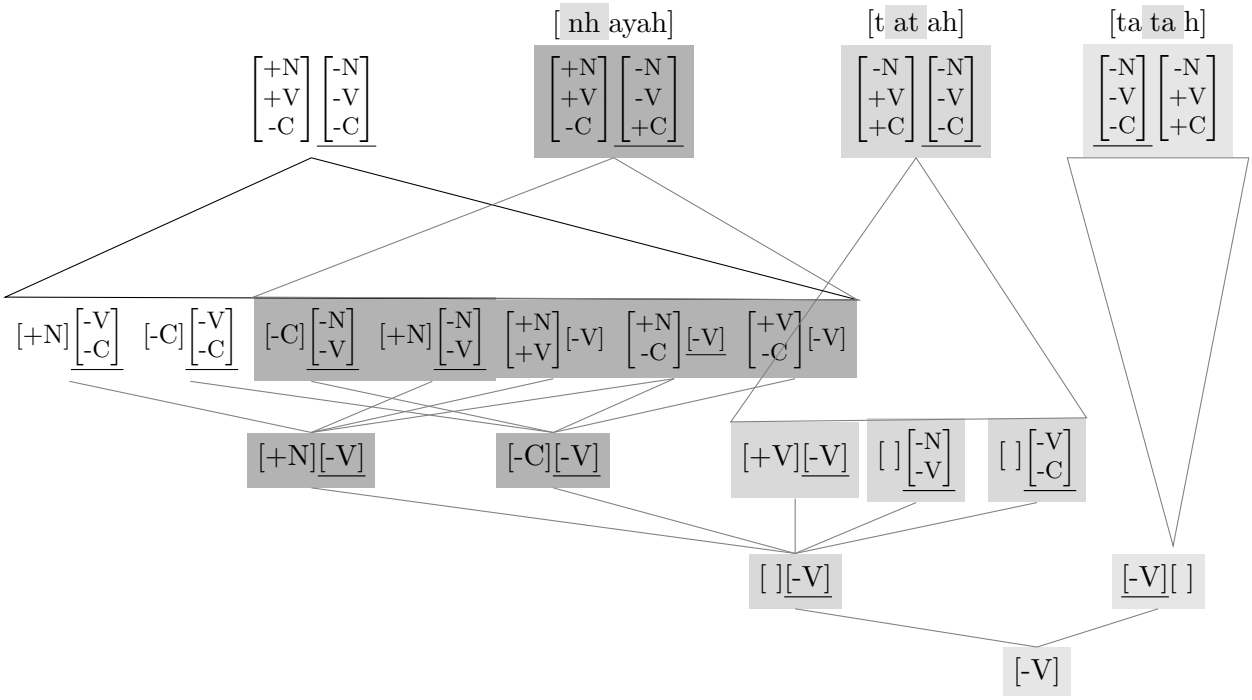


Figure 4.17: Updated hypothesis space after observing [nhayah]. The principle filter ↓[+N,+V,-C][-N,-V,+C] is removed from $\mathcal{H}(\psi_{[voi].})$

(43) *Hypothesis updates given /ku?tpa/→[ku?tpa]*

$\mathcal{H}(\psi_{[voi]})$	Figure 4.18	
$min(\mathcal{H}(\psi_{[voi]}))$	$\{[+N][-V,-C]\}$	
$\psi_{[voi]}(x)$	$([nas](px) \wedge \neg[cont](x))$	
BMRS program	$[nas]'(x)$	$=$ if $[nas](x)$ then $\neg\bot$ else $\bot$
	$[voi]'(x)$	$=$ if $[voi](x)$ then $\neg\bot$ else $([nas](px) \wedge \neg[cont](x))$
	$[cont]'(x)$	$=$ if $[cont](x)$ then $\neg\bot$ else $\bot$
Phonological map	$[-voi,-cont] \rightarrow [+voi]/[+nas]\_\_$	

At this point, we have arrived at a hypothesis that explains all the data in (39): [-voi,-cont] sounds becomes [+voi] whenever they are preceded by a [+nas] sound. The BMRS program in (43) moreover models exactly this function. Since this is the most general hypothesis that accounts for voicing in Zoque, further updates of the sets $V(\psi_{[voi]})$ and $\hat{V}(\psi_{[voi]})$ will not change the hypothesis. The hypothesis updates are summarized in Figure 4.19.

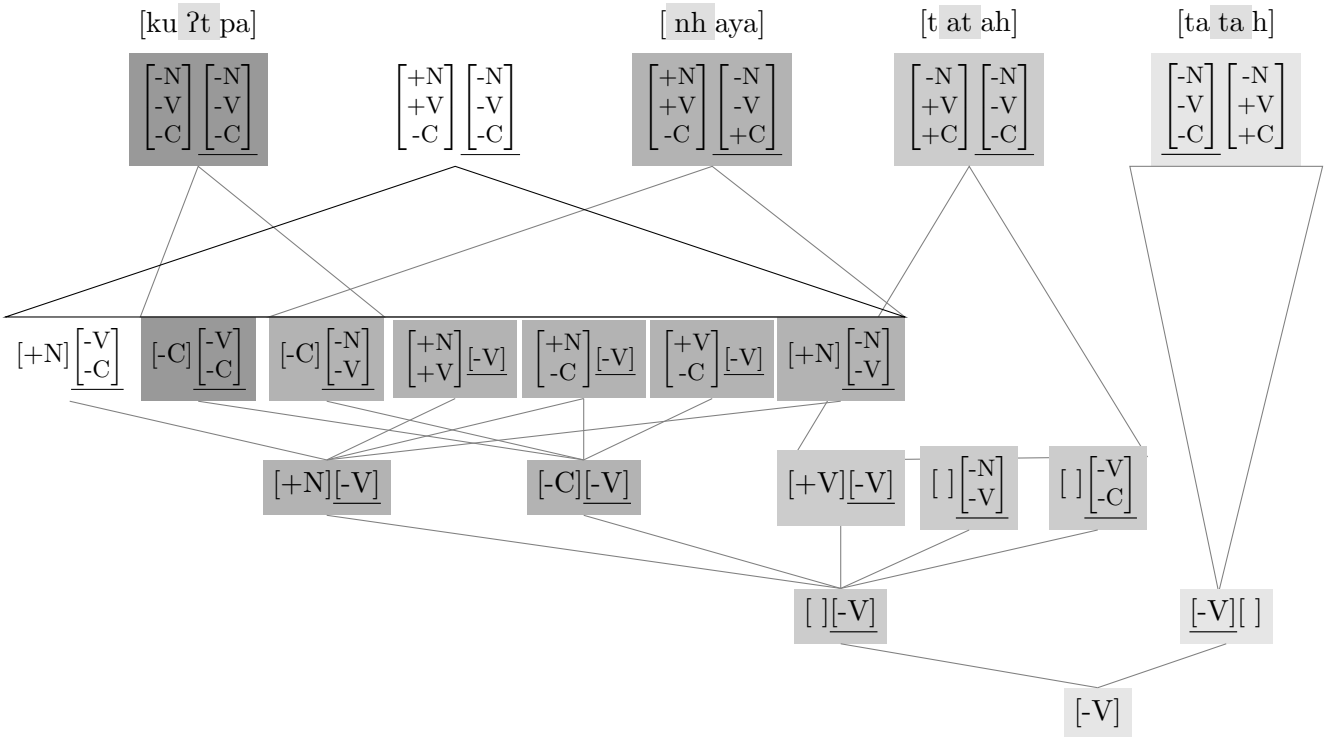


Figure 4.18: Updated hypothesis space after observing [ku?tpa]. The principle filter ↓[-N,-V,-C][-N,-V,-C] is removed from the hypothesis space.

The remaining subfactors in Figure 4.18 (i.e. those between the top element [+N,+V,-C][-N,-V,-C] and the minimal element [+N][-V,-C]) are presented in Figure 4.20. An interesting observation is that the subfactors in this space are exactly the set of ungrammatical structures that were presented in Figure 4.9. The only exception is these structures encode *environments*. In Figure 4.9, the space represents ungrammatical 2-factors for surface forms in Zoque, while Figure 4.20 represents the 2-factor *targets* for voicing. In Figure 4.9 [+N][-V,-C] was the most general description of ungrammatical 2-factors in Zoque. Because ungrammaticality is upward entailing, all the structures above [+N][-V,-C] are ungrammatical. In Figure 4.20, [+N][-V,-C] is the most general description of the environment where a sound becomes voiced. Because being a target for a phonological map is upward entailing, all the structures above [+N][-V,-C] are environments where voicing takes place. This observation highlights a concrete connection between surface constraints and phonological maps: *the substructures where voicing takes place are exactly the ungrammatical ones.* In other words, a [-V,-C] segment becomes voiced when it is preceded by a nasal as a means of *repairing* the fact that [+N][-V,-C] is an ungrammatical surface form. The example of postnasal voicing therefore

Sample	$Min(\mathcal{H}(\psi_{[voi]}))$	Hypothesis (as a map)
(mpama, mbama)	$\{[-V]\}$	$[-voice] \rightarrow [+voice]$
(tatah, tatah)	$\{[+N]\underline{[-V]}, [-C]\underline{[-V]}\}$	$[-voice] \rightarrow [+voice] \;/\; \begin{Bmatrix}[+nas]\\[-cont]\end{Bmatrix} \_\_$
(nhayah, nhayah)	$\{[+N]\underline{\begin{bmatrix}-V\\-C\end{bmatrix}}, [-C]\underline{\begin{bmatrix}-V\\-C\end{bmatrix}}\}$	$[-voice, -cont] \rightarrow [+voice] \;/\; \begin{Bmatrix}[+nas]\\[-cont]\end{Bmatrix} \_\_$
(kuʔtpa, kuʔtpa)	$\{[+N]\underline{\begin{bmatrix}-V\\-C\end{bmatrix}}\}$	$[-voi, -cont] \rightarrow [+voi]/[+nas]\_\_$

Figure 4.19: Summary of incremental hypothesis updates in (40)-(43).

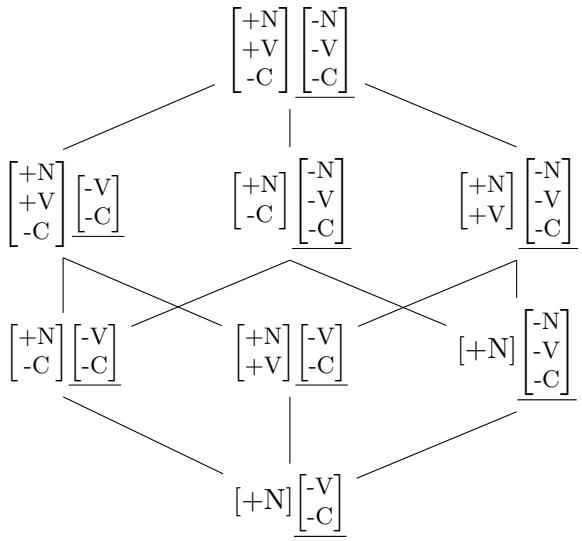


Figure 4.20: Remaining hypothesis in Figure 4.18. Voicing takes place at all of the 2-factors above [+N][-V,-C]. This poset looks exactly like the set of ungrammatical 2-factors in Figure 4.9.

shows how the same structure used for learning surface constraints can be adapted to learning maps. In this way, what the learning procedure presented in this chapter proposes is a means of learning *phonological generalizations*. In the case of Zoque, [-voi,-cont] sounds undergo voicing when they are preceded by a [+nas] sound because [+nas][-voi,-cont] structures are ungrammatical in Zoque. The hypothesis space and the corresponding BMRS program encode exactly this relationship.

4.4 Discussion

This chapter ultimately showed that model-theoretic approaches to learning stringsets can be adapted to learning maps by representing maps as logical transductions between string models. This chapter showed how the language of BMRS makes the jump from learning stringsets to learning maps possible, and does so in a way that represents phonological generalizations. While this work is very preliminary with many simplifying assumptions, there are promising ways to enrich the

hypothesis space and extend the types of processes that can be learned through this perspective.

The ISL functions are ones in which there is a maximum bound on how far we need to check to the left or right of the input in order to determine an output. The reason these functions can be expressed in the canonical normal form defined in Section 4.1.1 is because the information needed to determine the surface form of any sound is a fixed distance away in the input and can therefore be expressed with a finite disjunction of atomic expressions. In the case of $f : a \rightarrow b/b\_\_$, for example, the logical formula which expresses when an 'a' input becomes a non-'a' output is simply $\phi_a(x) = P_b(px)$. By contrast, recall the long-distance (unbounded) function $f^* : a \mapsto b/bx^*\_\_$. A formula which describes the environment in which an 'a' input becomes a non-'a' output would require an infinite disjunction: $\phi_a(x) = P_b(px) \vee P_b(ppx) \vee \cdots \vee P_b(p^n x) \vee \ldots$. It is for this reason that the program which models f^* in (12) of Section 2.2.2 requires an auxiliary recursive predicate `b-left` that allows the program to look an unbounded distance to the left.

The OSL functions, on the other hand, are an interesting in-between case; they require recursion, but the information needed to determine the output is within a window of fixed size. Consider again the OSL function $f_{it} : a \rightarrow b/b\_\_(iterate)$ in (9) of Section 2.2.2. This function requires local information in the *output*. For that reason, the environment in which an 'a' input becomes a non-'a' output is $\phi_a(x) = P_{it}(px)$. An interesting question for future work is what class of functions CNF-BMRS^{p,s} models if we allow both input and output predicates in the DNF formulas ϕ_σ and ψ_σ. In the case of f_{it}, the original BMRS program which models this function is repeated in (44) with equivalent CNF expressions for both $P_a^{f_{it}}$ and $P_b^{f_{it}}$, where $\phi_a(x)$ and $\psi_b(x)$ are highlighted. Intuitively, every OSL function should be expressible in this syntactic form because every OSL function is defined explicitly in terms of input and output predicates, and no auxiliary recursive functions, as discussed in Section 3.2. The significance of this observation is that it may be possible to extend the learning procedure in this chapter to OSL functions simply by adding reference to both input and output substructures in the hypothesis space.

(44) *The function* $f_{it} : a \to b/b\_\_(iterate)$ *as a CNF BMRS program*

$$
\begin{aligned}
P_a^{f_{it}}(x) &= \text{if } P_b^{f_{it}}(p(x)) \text{ then } \bot \text{ else } P_a(x) \\
&\equiv \text{if } P_a(x) \text{ then } \neg\, P_b^{it}(x) \text{ else } \bot \\
P_b^{f_{it}}(x) &= \text{if } P_a(x) \text{ then } P_b^{f_{it}}(p(x)) \text{ else } P_b(x) \\
&\equiv \text{if } P_b(x) \text{ then } \neg\bot \text{ else } (P_a(x) \wedge P_b^{it}(px))
\end{aligned}
$$

A further way to enrich the structures in the hypothesis space is to add ordering relations beyond just the predecessor and successor functions. It was briefly discussed in Section 2.1 that string models can be enriched with ordering relations that allow for representation of tiers, syllables, and prosodic structure. A much larger project on model-theoretic learning of phonological maps would be to learn maps over these complex representations.

Because the purpose of this chapter was to propose a first approach to learning phonological maps within the BMRS framework, we relied on several simplifying assumptions. First, this chapter only considered length-preserving transformations. Second, we assumed a trivial correspondence between the input and output indices. This means that the segment at index x in the output string is the surface form of the segment at index x in the input string. These assumptions rule out deletion, epenthesis, and metathesis maps which are also ISL [Chandlee, 2014]. An interesting extension of this work would be to develop a learning procedure for input-output index correspondences. Consider for example a simple deletion map $f : a \to \epsilon/b\_\_b$, where an 'a' is deleted whenever it is between two 'b's. For example, $f : ababa \mapsto abba$. Figure 4.21 presents some possible correspondences between input and output indices. The correct correspondence for the deletion map is in (b). These correspondences between input and output segments are called origin maps [Bojańczyk, 2014, Dolatian et al., 2021b]. Learning the map f would entail learning that the *correct* origin map, along with the environment that triggers 'a'-deletion.

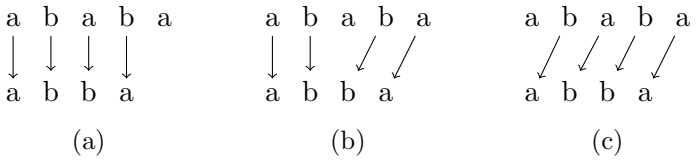


Figure 4.21: Possible correspondences between input and output segments (i.e. origin maps) in the case of deletion, where $ababa \mapsto abba$.

5

COMPOSITION AND DECOMPOSITION OF LOGICAL TRANSDUCTIONS

Phonological grammars contain collection of ordered maps; for each underlying form, the surface form is the application of the composition of these maps. The challenge of learning compositions is inferring the individual maps that are composed to yield the surface form of a language. This chapter considers the simplified situation in which the individual phonological maps in a grammar do not *interact*. In this case, the learning procedure in Chapter 4 yields a *syntactic* solution to the problem of learning the individual maps in a non-interacting composition. This chapter shows how to model the composition and simultaneous application of string functions as BMRS programs, and defines non-interacting compositions, and lays the formal background for showing how we can learn the *composition* of functions from a sample of input-output pairs when those functions are non-interacting. Section 5.1 shows how to model composition and simultaneous application as operators over BMRS program. Section 5.2 defines 'non-interacting' as a property BMRS compositions, and shows that non-interacting compositions can equivalently be expressed as simultaneous application. Section 5.3 demonstrates the significance of this result by showing that it yields a way of learning non-interacting compositions of phonological maps.

5.1 Operators over Programs

Two types of operators over string models are of significance to phonology: composition and simultaneous application. A phonological grammar is a *collection* of phonological maps from underlying representations to surface (phonetic) representations. A significant question for phonological theory is how the phonological maps yield surface representations. In particular, are these maps ordered with respect to each other, or do they all to apply underlying representations in parallel? The Direct Mapping Hypothesis [Chomsky and Halle, 1968, Kenstowicz and Kisseberth, 1979] posits that phonological maps apply directly to the underlying form (input) rather than in a sequential order. In other words, all the maps in a phonological grammar apply directly to the input if the environment for them to apply is found in the input string. This means that the surface form of each underlying representation is the result of the *simultaneous application* of the individual phonological maps in the grammar. As Kenstowicz and Kisseberth [1979] explain:

> The essential claim of the **Direct Mapping Hypothesis** is that the input structure to each of the phonological rules is the underlying representation (UR). The applicability of a rule is entirely a function of whether the underlying form meets the input requirements of that rule. The phonetic representation (PR) is then the result of applying the changes called for by the rule to the UR. The effects of a rule are necessarily irrelevant to the applicability of any other rule, since only the underlying form determines whether a rule applies or not. This view claims, then, that URs can be mapped into PRs without postulating intermediate levels of representation. [Kenstowicz and Kisseberth, 1979, pg.292]

Simultaneous application differs from *sequential ordering* of maps, in which the phonological maps in a grammar are ordered. This means that the first process in the sequence applies to the input to produce an intermediate representation; each subsequent map in the sequence applies to the output of the previous map. The surface forms of underlying representations in a language are then the *composition* of the individual maps. This is referred to as the Ordered-Rule Hypothesis:

> The **Ordered-Rule Hypothesis** maintains that phonological rules of the language are applied in sequence. The first rule operators on the underlying representation. Each subsequently applied rule operates on the structure resulting from the application of the previous rule. Furthermore, it maintains that the choice of what sequence to apply the rules in is not free. [...] Unlike direct mapping the ordered-rule hypothesis does not require that underlying representations be mapped to [surface] representations. It permits intermediate states, so the applicability of some rules can be determined by the applicability of other rules." [Kenstowicz and Kisseberth, 1979, pg.313-314]

The distinction between maps applying sequentially versus simultaneously is illustrated in Figures 5.1 and 5.2 using the three maps in (5.1). The function f^* was previously presented in (8) in Section 2.2.2. We consider first the composition of f^* and g^*, which yields a bidirectional function where an 'a' outputs as 'b' if it is *either preceded or followed* by a 'b'. The composition $f \circ g$ computes g^* on the input first and produces the intermediate form 'bbbaa' in Figure 5.1(a). The function f^* then applies to this intermediate form, rather than to the input to produce the output 'bbbbbb'. The same input-output relation between the input string 'aabaa' and the output string 'bbbbb' can be expressed as the simultaneous application of these two processes in Figure 5.1(b). In this case both f^* and g^* apply directly to the input, *without an intermediate representation.*

$$\begin{cases} f^*: & a \to b / bx^*\_\_ \\ g^*: & a \to b / \_\_x^* b \\ h: & b \to c / a\_\_a \end{cases} \tag{5.1}$$

x	a	a	b	a	a
	$g^*\downarrow$	$g^*\downarrow$			
$g^*(x)$	b	b	b	a	a
				$f^*\downarrow$	$f^*\downarrow$
$(f^* \circ g^*)(x)$	b	b	b	b	b

(a) Composition of f^* and g^*: g^* applies to the input string, and f^* applies to the output of g^*.

x	a	a	b	a	a
	$g^*\downarrow$	$g^*\downarrow$		$f^*\downarrow$	$f^*\downarrow$
$(f^* \mathbin{\textcircled{v}} g^*)(x)$	b	b	b	b	b

(b) Simultaneous application of f^* and g^*: both functions apply directly to the input.

Figure 5.1: Difference between composition and simultaneous application of the two functions f^* and g^* in (5.1). In general, $f^* \circ g^* = f^* \mathbin{\textcircled{v}} g^*$ on all strings.

The composition of $f^* \circ g^*$ is equivalent to the simultaneous application $f^* \mathbin{\textcircled{v}} g^*$ for all strings. This is in general not the case for the simultaneous application of any two functions. Consider the two functions g^* and h from (5.1) over the input string 'aabaa'. If g^* applies first, it will yield the output 'bbbaa'. In this case, the environment for h is not present in the intermediate representation, so h cannot not make any further changes. This is presented in Figure 5.2(a). In the reverse order, if h applies first, it will yield the output 'aacaa'. Now the environment for g^* is not present. Each of the processes g^* and h blocks the application of the other. Sequential ordering of processes therefore yields two different outputs depending on the order in which they apply. However, the environments for g^* and h are both present in the input representation. The

[0]Note that this concept differs from 'simultaneous application' in Johnson [1972], which refers to a single phonological rule applying simultaneously to all segments in the string, as opposed to iterative application.

simultaneous application of these two functions applies them both directly to the input, yielding the output string 'bbcaa', as in Figure 5.2(b).

x	a	a	b	a	a
	$g^*\downarrow$	$g^*\downarrow$			
$g^*(x)$	b	b	b	a	a
$(h \circ g^*)(x)$	b	b	b	a	a

(a) Composition of g^* and h: application of g^* removes the environment for h to apply.

x	a	a	b	a	a
	$g^*\downarrow$	$g^*\downarrow$	$h\downarrow$		
$(h \between g^*)(x)$	b	b	c	a	a

(b) Simultaneous application of g^* and h: both functions apply directly to the input.

Figure 5.2: Difference between composition and simultaneous application of two functions g^* and h. The string 'aabaa' provides evidence that $h \circ g^* \neq h \between g^*$.

Recall that a string function $f : \Sigma^* \to \Sigma^*$ is represented as a map π^f from Σ-structures to Σ-structures such that for every $w \in \Sigma^*$ and corresponding model $\mathcal{M}(w)$, the map π^f yields a structure $\mathcal{M}(f(w))$ (Section 2.1.3). That is, for every word w and string model of w, the transformation π^f gives a string model of $f(w)$. In this way, π^f represents the string function f as a relation between string models. This section defines operators over these maps that represents the *composition* and *simultaneous application* of two string functions as a relation on string models. Given *two* functions $f, g : \Sigma^* \to \Sigma^*$, and two maps $\pi^f, \pi^g : \mathit{Struc}(\Sigma) \to \mathit{Struc}(\Sigma)$, the goal is to define define an operators $\circ$ and $\between$ over π^f and π^g such that for every word $w \in \Sigma^*$ and corresponding model $\mathcal{M}(w)$, $\pi^f \circ \pi^g$ yields the model $\mathcal{M}(f \circ g(w))$ and $\pi^f \between \pi^g$ yields the model $\mathcal{M}(f \between g(w))$.

The remainder of this section formally defines the composition and simultaneous application of two maps as operators over BMRS programs. We assume that two programs F and G are defined over the same alphabet (and signature) Σ; for every input predicate P_σ, F and G have corresponding output predicates P^f_σ and P^g_σ respectively. For simplicity, we focus only on length-preserving maps (i.e. those without copy sets).

5.1.1 Composition

The composition operator that is defined over two BMRS programs here is identical to the operator $\otimes$ in Oakden [2021]. The definition given here differs in presentation, but captures exactly the same idea and effectively the same operator. The discussion of composition in this section is short; more detailed discussion of function interaction and the significance of $\otimes$ for modeling phonological rule

interaction can be found in Chapter 5 of Oakden [2021].

Definition 5.1 (**Composition of (Length-Preserving) BMRS Programs**)**.** (adapted from Oakden [2021]) For an input signature Σ with predicates $\{P_\sigma\}_{\sigma\in\Sigma}$, let $F = \{P_\sigma^f, \vec{f}\}_{\sigma\in\Sigma}$ and $G = \{P_\sigma^g, \vec{g}\}_{\sigma\in\Sigma}$ be programs over Σ, where $\vec{f}$ and $\vec{g}$ are collections of auxiliary recursive functions. For each $\sigma \in \Sigma$, the composition of P_σ^f and P_σ^g is defined as

$$P_\sigma^{f\circ g}(x) := P_\sigma^f\big[P_\sigma^g, P_\sigma^{f\circ g}/P_\sigma, P_\sigma^f\big](x) \tag{5.2}$$

The composition of F and G is defined as

$$F \circ G := \{P_\sigma^{f\circ g}, \vec{f}\}_{\sigma\in\Sigma} \cup G$$

For a model $\mathcal{M} = \langle D, P_\sigma, p, s\rangle_{\sigma\in\Sigma}$ and two maps π^f and π^g where

$$\pi^f : \mathcal{M} \to \langle D, P_\sigma^f, s, p\rangle_{\sigma\in\Sigma}$$
$$\pi^g : \mathcal{M} \to \langle D, P_\sigma^g, s, p\rangle_{\sigma\in\Sigma}$$

the composition of π^f and π^g is the following map over the input model:

$$\pi^f \circ \pi^g : \mathcal{M} \to \langle D, P_\sigma^{f\circ g}, s, p\rangle_{\sigma\in\Sigma}$$

The equation in (5.2) says that the composition $P_\sigma^{f\circ g}$ is the result of replacing all occurrences of P_σ in the definition of P_σ^f with P_σ^g, and replacing all occurrences of P_σ^f with $P_\sigma^{f\circ g}$. This means that whenever the outer function P_σ^f refers to the input predicate P_σ, the composition will instead refer to the *output* of the inner function P_σ^g. Similarly, whenever the outer function P_σ^f recursively refers to itself, the composition $P_\sigma^{f\circ g}$ will recursively refer to itself. Oakden [2021] shows how this syntactic definition of composition works for ISL (non-recursive) maps. Example 5.2 presents a quick example with two ISL functions. Example 5.3 presents a quick example with an iterative recursive function.

Example 5.2 (Composition with Non-Recursive Functions)**.** This example presents the composition of two string functions $f : a \to b/b\_\_$ and $g : c \to a/\_\_c$ as a composition of their respective BMRS programs. The BMRS program which models f was presented in (6) in Section 2.2.1, and repeated below. The BMRS program which models g is presented in (45).

(6) *The function* $f : a \rightarrow b/b\_\_$ *as a BMRS program* $F = \{P_a^f, P_b^f, P_c^f\}$

$$P_a^f(x) = \text{if } P_b(p(x)) \text{ then } \bot \text{ else } P_a(x)$$
$$P_b^f(x) = \text{if } P_a(x) \text{ then } P_b(p(x)) \text{ else } P_b(x)$$
$$P_c^f(x) = P_c(x)$$

(45) *The function* $g : c \rightarrow a/\_\_c$ *as a BMRS program* $G = \{P_a^g, P_b^g, P_c^g\}$

$$P_a^g(x) = \text{if } P_c(x) \text{ then } P_c(s(x)) \text{ else } P_a(x)$$
$$P_b^g(x) = P_b(x)$$
$$P_c^g(x) = \text{if } P_c(s(x)) \text{ then } \bot \text{ else } P_c(x)$$

The composition $f \circ g$ is modeled by the composition of the two programs F and G in (46). The resulting program is exactly F, with all occurrences of input predicates $\{P_a, P_b, P_c\}$ being replaced by the output predicates of the inner program: $\{P_a^g, P_b^g, P_c^g\}$. Figure 5.3 compares the programs F, G, and $F \circ G$ over the input string model for 'bcc'.

(46) *The function* $f \circ g$ *as the composition* $F \circ G = \{P_a^{f\circ g}, P_b^{f\circ g}, P_c^{f\circ g}\}$

$$P_a^{f\circ g}(x) = \text{if } P_b^g(p(x)) \text{ then } \bot \text{ else } P_a^g(x)$$
$$P_b^{f\circ g}(x) = \text{if } P_a^g(x) \text{ then } P_b^g(p(x)) \text{ else } P_b^g(x)$$
$$P_c^{f\circ g}(x) = P_c^g(x)$$

	b	c	c
$P_a(x)$	$\bot$	$\bot$	$\bot$
$P_b(x)$	$\top$	$\bot$	$\bot$
$P_c(x)$	$\bot$	$\top$	$\top$
$P_a^f(x)$	$\bot$	$\bot$	$\bot$
$P_b^f(x)$	$\top$	$\bot$	$\bot$
$P_c^f(x)$	$\bot$	$\top$	$\top$
	b	c	c

(a) F

	b	c	c
$P_a(x)$	$\bot$	$\bot$	$\bot$
$P_b(x)$	$\top$	$\bot$	$\bot$
$P_c(x)$	$\bot$	$\top$	$\top$
$P_a^g(x)$	$\bot$	$\top$	$\bot$
$P_b^g(x)$	$\top$	$\bot$	$\bot$
$P_c^g(x)$	$\bot$	$\bot$	$\top$
	b	a	c

(b) G

	b	c	c
$P_a^g(x)$	$\bot$	$\top$	$\bot$
$P_b^g(x)$	$\top$	$\bot$	$\bot$
$P_c^g(x)$	$\bot$	$\bot$	$\top$
$P_a^{f\circ g}(x)$	$\bot$	$\bot$	$\bot$
$P_b^{f\circ g}(x)$	$\top$	$\top$	$\bot$
$P_c^{f\circ g}(x)$	$\bot$	$\bot$	$\top$
	b	b	c

(c) $F \circ G$

Figure 5.3: Application of the program F and G versus the composition $F \circ G$ over the input model for 'bcc'. F and G are defined over the input model; $F \circ G$ is defined over the output values of G.

Example 5.3 (Composition with Iterative Function)**.** This example presents the composition of the iterative string function f_{it} from (9) of Section 2.2.2 (repeated below) with the function g from the previous example. The composition $f_{it} \circ g$ is modeled by the composition of the two programs F_{it} and G in (47). The resulting program is exactly F_{it}, with all occurrences of input predicates $\{P_a, P_b, P_c\}$ being replaced by the output predicates of the inner program: $\{P_a^g, P_b^g, P_c^g\}$, and all occurrences of the output predicate $P_b^{f_{it}}$ replaced by $P_b^{f_{it}\circ g}$ (highlighted).

(9) *The function* $f_{it} : a \to b/b\_(iterate)$ *as a BMRS program* $F_{it} = \{P_a^{f_{it}}, P_b^{f_{it}}, P_c^{f_{it}}\}$

$$P_a^{f_{it}}(x) = \text{if } P_b^{f_{it}}(p(x)) \text{ then } \bot \text{ else } P_a(x)$$
$$P_b^{f_{it}}(x) = \text{if } P_a(x) \text{ then } P_b^{f_{it}}(p(x)) \text{ else } P_b(x)$$
$$P_c^{f_{it}}(x) = P_c(x)$$

(47) *The composition* $f_{it} \circ g$ *as the composition* $F_{it} \circ G = \{P_a^{f_{it} \circ g}, P_b^{f_{it} \circ g}, P_c^{f_{it} \circ g}\}$

$$P_a^{f_{it} \circ g}(x) = \text{if } P_b^{f_{it} \circ g}(p(x)) \text{ then } \bot \text{ else } P_a^g(x)$$
$$P_b^{f_{it} \circ g}(x) = \text{if } P_a^g(x) \text{ then } P_b^{f_{it} \circ g}(p(x)) \text{ else } P_b^g(x)$$
$$P_c^{f_{it} \circ g}(x) = P_c^g(x)$$

Figure 5.4 compares the programs F and $F_{it} \circ G$ over the input string 'bacc'. Since the composition is recursive, we need to evaluate it using the least fixed point of monotone sequences of functions. Figure 5.4(a) presents the sequence of functions used to evaluate the recursive equation $P_b^{f_{it}}$; the iterative process stops at index 2, where the first non-'a' symbol is found. The details of how the sequence is generated and how the predicate is evaluated were discussed in Example 2.10 of Section 2.2.2. Figure 5.4(b) present the sequence for $P_b^{f_{it} \circ g}$; since $P_a^g(2)$ evaluates to $\top$, the iterative process goes up to index 2 in the composition.

	b	a	c	c
$b_0(x)$	!	!	!	!
$b_1(x)$	$\top$	!	$\bot$	$\bot$
$b_2(x)$	$\top$	$\top$	$\bot$	$\bot$

(a) Evaluating $P_b^{f_{it}}(x)$

	b	a	c	c
$P_a^g(x)$	$\bot$	$\top$	$\top$	$\bot$
$P_b^g(x)$	$\top$	$\bot$	$\bot$	$\bot$
$P_c^g(x)$	$\bot$	$\bot$	$\bot$	$\top$
$b_0(x)$	!	!	!	!
$b_1(x)$	$\top$	!	!	$\bot$
$b_2(x)$	$\top$	$\top$	!	$\bot$
$b_3(x)$	$\top$	$\top$	$\top$	$\bot$

(b) Evaluating $P_b^{f_{it} \circ g}(x)$

Figure 5.4: Monotone sequences of functions used to evaluate recursive predicate ${P_b}^{f_{it}}$ in F versus the predicate ${P_b}^{f_{it} \circ g}$ in $F \circ G$ over the input model for the string 'bacc'.

5.1.2 Simultaneous Application

The concept of simultaneous application of two string functions is as follows: for each x in the input string, if either f or g changes the value of x, the change is reflected in the output; otherwise, the input stays the same. In order to define simultaneous application over programs, we first define it over the individual predicates in programs. Simultaneous application of two output predicates is as follows: if either P_σ^f or P_σ^g flips the truth value of the input predicate P_σ, the simultaneous

application of P^f_σ and P^g_σ flips the truth value. Figure 5.5 shows how simultaneous application of two output predicates works. For any input predicate P_σ and any index x, if $P_\sigma(x) = \top$ then $P^{f \mathbin{\mathbb{O}} g}_\sigma(x) = \top$ exactly when both $P^f_\sigma(x) = \top$ and $P^g_\sigma(x) = \top$ (i.e. neither P^f_σ nor P^g_σ flips the truth value). Similarly, if $P_\sigma(x) = \bot$ then $P^{f \mathbin{\mathbb{O}} g}_\sigma(x) = \bot$ exactly when both $P^f_\sigma(x) = \bot$ and $P^g_\sigma(x) = \bot$. The highlighted regions of Figure 5.5 indicate when a truth value has flipped. Definition 5.4 gives the definition for simultaneous application defined over length-preserving BMRS programs. The BMRS equation in (5.3) computes exactly the truth table in Figure 5.5. The simultaneous application of two BMRS programs then applies this operator pointwise to each of the predicates, as in (**??**) for systems of equations and (**??**) for maps over Σ-models. Since all the maps discussed in this chapter are length-preserving, we only consider a simultaneous application operator for length-preserving programs. It is further possible to define an extension of simultaneous application to copy sets and metathesis maps, but this is outside the scope and goals of this dissertation.

$P_\sigma(x)$	$P^f_\sigma(x)$	$P^g_\sigma(x)$	$P^{f \mathbin{\mathbb{O}} g}_\sigma(x)$
$\top$	$\top$	$\top$	$\top$
$\top$	$\top$	$\bot$	$\bot$
$\top$	$\bot$	$\top$	$\bot$
$\top$	$\bot$	$\bot$	$\bot$
$\bot$	$\top$	$\top$	$\top$
$\bot$	$\top$	$\bot$	$\top$
$\bot$	$\bot$	$\top$	$\top$
$\bot$	$\bot$	$\bot$	$\bot$

Figure 5.5: Simultaneous application of two output predicates, as a truth table. If either $P^f(x)$ or $P^g(x)$ flips the truth value associated with $P(x)$, then $P^{f \mathbin{\mathbb{O}} g}(x)$ flips the truth value (highlighted).

Definition 5.4 (**Simultaneous Application of (Length-Preserving) BMRS Programs**)**.** For an input signature Σ with predicates $\{P_\sigma\}_{\sigma\in\Sigma}$, let $F = \{P^f_\sigma, \vec{f}\}_{\sigma\in\Sigma}$ and $G = \{P^g_\sigma, \vec{g}\}_{\sigma\in\Sigma}$ be programs over Σ, where $\vec{f}$ and $\vec{g}$ are collections of auxiliary recursive functions. For each $\sigma \in \Sigma$, the simultaneous application of P^f_σ and P^g_σ is defined as

$$P^{f \mathbin{\mathbb{O}} g}_\sigma(x) := \text{if } P_\sigma(x) \text{ then } \big(P^f_\sigma(x) \wedge P^g_\sigma(x)\big) \text{ else } \big(P^f_\sigma(x) \vee P^g_\sigma(x)\big) \tag{5.3}$$

The simultaneous application of F and G is defined as

$$F \mathbin{\mathbb{O}} G := \{P^{f \mathbin{\mathbb{O}} g}_\sigma\}_{\sigma\in\Sigma} \cup F \cup G$$

For a model $\mathcal{M} = \langle D, P_\sigma, p, s\rangle_{\sigma\in\Sigma}$ and two maps π^f and π^g where

$$\pi^f : \mathcal{M} \to \langle D, P^f_\sigma, s, p\rangle_{\sigma\in\Sigma}$$
$$\pi^g : \mathcal{M} \to \langle D, P^g_\sigma, s, p\rangle_{\sigma\in\Sigma}$$

the simultaneous application of π^f and π^g is the following map over the input model:

$$\pi^f \circledcirc \pi^g : \mathcal{M} \to \langle D, P_\sigma^{f \circledcirc g}, s, p \rangle_{\sigma \in \Sigma}$$

For any input model with predicates $\{P_\sigma\}_{\sigma\in\Sigma}$, the output predicates $\{P_\sigma^{f\circledcirc g}\}_{\sigma\in\Sigma}$ in the program $F \circledcirc G$ determine the output string. In other words, the string-to-string function that $F \circledcirc G$ models is encoded in the relationship between input predicates P_σ and their corresponding output predicates $P_\sigma^{f\circledcirc g}$. For two systems of equations F and G that model string-to-string functions, the simultaneous application $F \circledcirc G$ models the result of applying both functions to the input without an intermediate representation. Proposition 5.5 presents three algebraic properties of the simultaneous application operator that are immediate consequences of Definition 5.4.

Proposition 5.5 (Algebraic Properties)**.** For an input signature $\Sigma = \langle P_\sigma, p, s \rangle_{\sigma\in\Sigma}$ and programs $F = \{P_\sigma^f\}_{\sigma\in\Sigma}$, $G = \{P_\sigma^g\}_{\sigma\in\Sigma}$, and $H = \{P_\sigma^h\}_{\sigma\in\Sigma}$ over Σ, the simultaneous application operator has the following three properties.

(a) Commutativity. $F \circledcirc G = G \circledcirc F$

(b) Associativity. $(F \circledcirc G) \circledcirc H = F \circledcirc (G \circledcirc H)$

(c) Identity. The identity program $I_\Sigma = \{P_\sigma^I(x) = P_\sigma(x)\}_{\sigma\in\Sigma}$ is such that $F \circledcirc I_\Sigma = F$.

Proof. (Commutativity) It follows from commutativity of conjunction and disjunction that $P_\sigma^{f\circledcirc g} = P_\sigma^{g\circledcirc f}$ for every $\sigma \in \Sigma$, and therefore $F \circledcirc G = G \circledcirc F$.
(Associativity) $(F \circledcirc G) \circledcirc H$ is the system of equations $\{P_\sigma^{f\circledcirc g} \circledcirc P_\sigma^h\}_{\sigma\in\Sigma}$. By definition,

$$P_\sigma^{f\circledcirc g} \circledcirc P_\sigma^h(x) = \text{if } P_\sigma(x) \text{ then } (P_\sigma^{f\circledcirc g}(x) \wedge P_\sigma^h(x)) \text{ else } (P_\sigma^{f\circledcirc g}(x) \vee P_\sigma^h(x)) \tag{5.4}$$

If $P_\sigma(x) = \top$, then $P_\sigma^{f\circledcirc g}(x)$ will evaluate to $P_\sigma^f(x) \wedge P_\sigma^g(x)$ and $P_\sigma^{g\circledcirc h}(x)$ will evaluate to $P_\sigma^g(x) \wedge P_\sigma^h(x)$. If $P_\sigma(x) = \bot$, then $P_\sigma^{f\circledcirc g}(x)$ will evaluate to $P_\sigma^f(x) \vee P_\sigma^g(x)$ and $P_\sigma^{g\circledcirc h}(x)$ will evaluate to $P_\sigma^g(x) \vee P_\sigma^h(x)$. The following equivalences then follow from associativity of of conjunction and disjunction.

$$\begin{aligned} P_\sigma^{f\circledcirc g} \circledcirc P_\sigma^h(x) &= \text{if } P_\sigma(x) \text{ then } ((P_\sigma^f(x) \wedge P_\sigma^g(x)) \wedge P_\sigma^h(x)) \text{ else } ((P_\sigma^f(x) \vee P_\sigma^g(x)) \vee P_\sigma^h(x)) \\ &= \text{if } P_\sigma(x) \text{ then } (P_\sigma^f(x) \wedge P_\sigma^{g\circledcirc h}(x)) \text{ else } (P_\sigma^f(x) \vee P_\sigma^{g\circledcirc h}(x)) \\ &= P_\sigma^f \circledcirc P_\sigma^{g\circledcirc h} \end{aligned}$$

(Identity) Let I_Σ be the system of equations $\{P^{\mathtt{id}}_\sigma(x) = P_\sigma(x)\}_{\sigma\in\Sigma}$. When $P_\sigma(x) = \top$, $P^{\mathtt{id}}_\sigma(x) = \top$, so $P^f_\sigma(x) \wedge P^{\mathtt{id}}_\sigma(x)$ is logically equivalent to $P^f_\sigma(x)$. Similarly, when $P_\sigma(x) = \bot$, $P^{\mathtt{id}}_\sigma(x) = \bot$, so $P^f_\sigma(x) \vee P^{\mathtt{id}}_\sigma(x)$ is logically equivalent to $P^f_\sigma(x)$.

$$\begin{aligned} P^{f \mathbin{\mathbb{Q}} \mathtt{id}}_\sigma(x) &= \text{if } P_\sigma(x) \text{ then } P^f_\sigma \wedge P^{\mathtt{id}}_\sigma(x) \text{ else } P^f_\sigma \vee P^{\mathtt{id}}_\sigma(x) \\ &= \text{if } P_\sigma(x) \text{ then } P^f_\sigma(x) \text{ else } P^f_\sigma(x) \\ &= P^f_\sigma(x) \end{aligned}$$

□

We consider a few examples that show how Definition 5.4 succeeds in modeling the simultaneous application of two *length-preserving string functions*. Example 5.6 presents the simplest use of simultaneous application, with two non-recursive functions in which the composition and simultaneous application are equivalent. Example 5.7 shows what happens when two functions conflict over what output to produce. Example 5.8 revisits the two programs in Example 5.2 in order to compare composition and simultaneous. Example 5.9 shows how the simultaneous application of two iterative functions yields a bidirectional map.

Example 5.6. Recall the function $f : a \to b/b\_\_$ from Example 5.2. A similar function $g : a \to b/\_\_b$ is presented in (48) with the system of equations G. The simultaneous application of F and G is presented in (49); the two predicate $P^{f \mathbin{\mathbb{Q}} g}_a$ and $P^{f \mathbin{\mathbb{Q}} g}_b$ are identical since the syntactic operator over expressions in (5.3) is applied pointwise to the predicates in F and G.

(48) *The function $g : a \to b/\_\_b$ as a BMRS program $G = \{P^g_a, P^g_b, P^g_c\}$*

$$\begin{aligned} P^g_a(x) &= \text{if } P_b(s(x)) \text{ then } \bot \text{ else } P_a(x) \\ P^g_b(x) &= \text{if } P_a(x) \text{ then } P_b(s(x)) \text{ else } P_b(x) \\ P^g_c(x) &= P_c(x) \end{aligned}$$

(49) *Simultaneous application of the program F and G*

$$\begin{aligned} P^{f \mathbin{\mathbb{Q}} g}_a(x) &= \text{if } P_a(x) \text{ then } \big(P^f_a(x) \wedge P^g_a(x)\big) \text{ else } \big(P^f_a(x) \vee P^g_a(x)\big) \\ P^{f \mathbin{\mathbb{Q}} g}_b(x) &= \text{if } P_b(x) \text{ then } \big(P^f_b(x) \wedge P^g_b(x)\big) \text{ else } \big(P^f_b(x) \vee P^g_b(x)\big) \end{aligned}$$

In Figure 5.6, the programs F and G are applied to the input string 'aba'. The highlighted Boolean values indicate when an output predicate has flipped the truth value of the corresponding input predicate. Figure 5.6(a) shows how the system of equations F yields the output string 'abb'

and (b) shows how G yields the output string 'bba'. Figure 5.7 then compares the composition $F \circ G$ with the simultaneous application $F \circledcirc G$. Figure 5.7(b) in particular shows how every $F \circledcirc G$ is the combination of all the truth values that were flipped by F and G individually. In other words, the resulting output string is the compilation of the changes made by F and by G.

	a	b	a
$P_a(x)$	$\top$	$\bot$	$\top$
$P_b(x)$	$\bot$	$\top$	$\bot$
$P_c(x)$	$\bot$	$\bot$	$\bot$
$P_a^f(x)$	$\top$	$\bot$	$\bot$
$P_b^f(x)$	$\bot$	$\top$	$\top$
$P_c^f(x)$	$\bot$	$\bot$	$\bot$
	a	b	b

(a) F

	a	b	a
$P_a(x)$	$\top$	$\bot$	$\top$
$P_b(x)$	$\bot$	$\top$	$\bot$
$P_c(x)$	$\bot$	$\bot$	$\bot$
$P_a^g(x)$	$\bot$	$\bot$	$\top$
$P_b^g(x)$	$\top$	$\top$	$\bot$
$P_c^f(x)$	$\bot$	$\bot$	$\bot$
	b	b	a

(b) G

Figure 5.6: Systems of equations F and G applied to the input model for the string 'aba'. Instances where an output predicate flips the truth value associated with the corresponding input predicate are highlighted.

	a	b	a
$P_a^g(x)$	$\bot$	$\bot$	$\top$
$P_b^g(x)$	$\top$	$\top$	$\bot$
$P_c^g(x)$	$\bot$	$\bot$	$\bot$
$P_a^{f \circ g}(x)$	$\bot$	$\bot$	$\bot$
$P_b^{f \circ g}(x)$	$\top$	$\top$	$\top$
$P_c^{f \circ g}(x)$	$\bot$	$\bot$	$\bot$
	b	b	b

(a) $F \circ G$

	a	b	a
$P_a^{f \circledcirc g}(x)$	$\bot$	$\bot$	$\bot$
$P_b^{f \circledcirc g}(x)$	$\top$	$\top$	$\top$
$P_c^{f \circledcirc g}(x)$	$\bot$	$\bot$	$\bot$
	b	b	b

(b) $F \circledcirc G$

Figure 5.7: Composition $F \circ G$ and simultaneous application $F \circledcirc G$ applied to the input model for the string 'aba'. $F \circ G$ applies F to the output of G while $F \circledcirc G$ applies F and G directly to the input.

Example 5.7 (Conflicting Functions)**.** This example presents a case where the two functions f and g both apply in the same environment, but conflict in what output they produce. Both functions change an 'a' in the input when it is immediately preceded by a 'b', but $f : a \to b/b\_\_$ changes it to a 'b' while $g : a \to c/b\_\_$ changes it to a 'c'. The function g is modeled by the systems of equations in (50). Figure 5.8 demonstrates the outputs of F, G, and their simultaneous application over the string 'ba'. Figures 5.8(a,b) show how F produces the output 'bb' and G produces the output 'bc'. Figure 5.8(c) shows the simultaneous application of F and G. All the changes made by F and all the changes made by G must be changes made by the simultaneous application of F and G. The result is a system of equations where $b^{f \circledcirc g}$ and $c^{f \circledcirc g}$ must both evaluate to $\top$ for the final index. This means that 'a' must output as both a 'b' and a 'c'. In this case, the simultaneous application

operator yields a program where the output for 'ba' cannot be interpreted as a string.

(50) *The function* $g : a \to c/b\_$ *as the BMRS program* $G = \{P_a^g, P_b^g, P_c^g\}$

$P_a^g(x) = \text{if } P_b(p(x)) \text{ then } \bot \text{ else } P_a(x)$

$P_b^g(x) = P_b(x)$

$P_c^g(x) = \text{if } P_a(x) \text{ then } P_b(p(x)) \text{ else } P_c(x)$

	b	a
$P_a^f(x)$	$\bot$	$\bot$
$P_b^f(x)$	$\top$	$\top$
$P_c^f(x)$	$\bot$	$\bot$
	b	b

(a) F

	b	a
$P_a^g(x)$	$\bot$	$\bot$
$P_b^g(x)$	$\top$	$\bot$
$P_c^g(x)$	$\bot$	$\top$
	b	c

(b) G

	b	a
$P_a^{f \between g}(x)$	$\bot$	$\bot$
$P_b^{f \between g}(x)$	$\top$	$\top$
$P_c^{f \between g}(x)$	$\bot$	$\top$
	b	!

(c) $F \between G$

Figure 5.8: Two conflicting systems of equations F and G. The predicates ${P_b}^f$ and ${P_c}^g$ evaluate to $\top$ for the final index (highlighted). Thus, the simultaneous application results in a string where the final index must be both a 'b' and a 'c', yielding a program that cannot be interpreted as a string.

In general, the syntax of BMRS does not explicitly rule out programs which are uninterpretable as strings for certain inputs. For string models, a well-defined transduction is one in which the input and output structures are both string models. In particular, this means that exactly one output predicate holds at each index. This condition of string models was given in Definition 2.3. Thus, the structure in Figure 5.8(c) is not a string model. The problem of conflicting programs (and well-definedness) looks different for feature (i.e. unconventional string) models than it does for string models. $F \between G$ cannot produce an output for the input string 'ba' because 'a' would need to output as both a 'b' and a 'c'. It is, however, possible for feature models because more than one feature can be true at each index. Since the primary focus of this dissertation is phonological models, this issue does not come up and is not discussed any further.

Example 5.8. This example shows how simultaneous application differs from composition. We reconsider the two functions f and g and corresponding programs F and G from Example 5.2. The composition of these two programs is repeated in Figure 5.9(a). In the case of simultaneous application, the program $F \between G$ is the collection of the changes made by F and by G to the input.

Example 5.9 (Iterative Functions)**.** This example presents the simultaneous application of the functions f_{it} and g_{it} in (5.1). The BMRS program F for the function f_{it} was presented in Example

	b	c	c
$P_a^g(x)$	⊥	⊤	⊥
$P_b^g(x)$	⊤	⊥	⊥
$P_c^g(x)$	⊥	⊥	⊤
$P_a^{f\circ g}(x)$	⊥	⊥	⊥
$P_b^{f\circ g}(x)$	⊤	⊤	⊥
$P_c^{f\circ g}(x)$	⊥	⊥	⊤
	b	b	c

(a) $F \circ G$

	b	c	c
$P_a^f(x)$	⊥	⊥	⊥
$P_b^f(x)$	⊤	⊥	⊥
$P_c^f(x)$	⊥	⊤	⊤
$P_a^g(x)$	⊥	⊤	⊥
$P_b^g(x)$	⊤	⊥	⊥
$P_c^g(x)$	⊥	⊥	⊤
$P_a^{f\owedge g}(x)$	⊥	⊤	⊥
$P_b^{f\owedge g}(x)$	⊤	⊥	⊥
$P_c^{f\owedge g}(x)$	⊥	⊥	⊤
	b	a	c

(b) $F \owedge G$

Figure 5.9: Composition $F \circ G$ and simultaneous application $F \owedge G$ applied to the input model for the string 'bcc'.

5.3. The program G for the function f_{it} is presented in (51). The monotone sequences of functions for the two predicates $\{P_a^{f\owedge g}, P_b^{f\owedge g}\}$ are presented in Figure 5.10.

(51) *The function $g : a \rightarrow b$/__b(iterate) as the BMRS program G*

$$P_a^g(x) = \text{if } P_b^g(s(x) \text{ then } \bot \text{ else } P_a(x)$$

$$P_b^g(x) = \text{if } P_a(x) \text{ then } P_b^g(s(x)) \text{ else } P_b(x)$$

	a	a	b	a	a
$a_1^{f\owedge g}(x)$	!	!	!	!	!
$b_1^{f\owedge g}(x)$	!	!	⊤	!	!
$a_2^{f\owedge g}(x)$	!	⊥	!	⊥	!
$b_2^{f\owedge g}(x)$	!	⊤	⊤	⊤	!
$a_3^{f\owedge g}(x)$	⊥	⊥	⊥	⊥	⊥
$b_3^{f\owedge g}(x)$	⊤	⊤	⊤	⊤	⊤
	b	b	b	b	b

Figure 5.10: Monotonic sequence of functions for the two output predicates in $F \owedge G$, illustrating how the simultaneous application of two iterative processes yields a bidirectional harmony process.

Examples of simultaneous application over phonological models are discussed in Chapter 6 in order to show how simultaneous application can be used to characterize the weakly deterministic functions. Figure 5.10, in particular, shows how the simultaneous application can be used to represent bidirectional processes which will be discussed further in Section 6.2.

5.2 Non-Interacting Composition

This section provides the main formal result that makes the connection between the composition and simultaneous application operators: non-interacting function composition can be expressed as simultaneous application. The starting point for defining non-interacting programs is the definition of non-interacting string functions proposed by McCollum et al. [2018]. The definition uses the concept of *mutation points*. For a general function f, the mutation points of f are the subset of elements in $dom(f)$ defined in (5.5).

$$\mu(f) := \{x \in dom(f) \mid f(x) \neq x\} \tag{5.5}$$

The mutation points of a map are the elements of the domain which undergo some change under the application of f. This concept is ultimately relevant for defining non-interacting compositions because we need to formally express when a string function/phonological map yields a change between the underlying and surface representations. For *string* functions $f : \Sigma^* \to \Sigma^*$, the mutation points of f are defined relative to a particular word in the following way: for each word $w \in \Sigma^*$, the mutation points of f are the indices of w which undergo change under the application of f, as in (5.6). Recall that w_i refers to the i^{th} element of the string w. For simplicity, we only consider length-preserving maps (i.e. $|w| = |f(w)|$) with a trivial one-to-one correspondence between input and output indices (i.e. index i of the input string maps to index i of the output string). Definition 5.10 gives the definition of mutation points for a string function, along with the relevant concept of μ-conserving. These concepts are discussed by Meinhardt et al. [2021], and revisited in Chapter 6.

Definition 5.10. Given an alphabet Σ, function $f : \Sigma^* \to \Sigma^*$, and word $w \in \Sigma^*$, the **mutation points** of f relative to w are the following set of indices of w.

$$\mu(f, w) = \{i < |w| \mid [f(w)]_i \neq w_i\} \tag{5.6}$$

Given functions $f, g : \Sigma^* \to \Sigma^*$, the composition $f \circ g$ is **μ-conserving** iff for all $w \in \Sigma^*$,

$$\mu(f \circ g, w) = \mu(g, w) \cup \mu(g, w)$$

Given this definition of a mutation point, Definition 5.11 gives an adaptation of McCollum et al. [2018]'s definition of non-interacting compositions of two *string* functions. This definition

essentially says that f and g are non-interacting when composed as $f \circ g$ if: (i) for any change made by the outer function f, the inner function g does not block the change, and (ii) for any change made by the inner function g, the outer function f does not make any additional changes. The definition in 5.11 deviates slightly from the definition proposed by McCollum et al. [2018], but captures then intended purpose.[1]

Definition 5.11 (adapted from McCollum et al. [2018])**.** Two string functions $f : \Sigma^* \to \Sigma^*$ and $g : \Sigma^* \to \Sigma^*$ are non-interacting when composed as $f \circ g$ iff for every $w \in \Sigma^*$ the following two conditions hold:

(i) for all $i \in \mu(f, w)$, $[(f \circ g)(w)]_i = [f(w)]_i$

(ii) for all $i \in \mu(g, w)$, $[(f \circ g)(w)]_i = [g(w)]_i$

Before providing the equivalent BMRS counterparts for these definitions, we consider what it means for two functions to *interact*. The pair of functions f and g in (52) illustrate conditions in (i) and (ii) of Definition 5.11 over the string 'bba'. (52a) shows that $f \circ g$ violates condition (i), and (52b) shows that $f \circ g$ violates condition (ii). The pair of functions f and g from Example 5.2 are revisited in (53) to show how they violate the condition (ii) of Definition 5.11.

(52) *Interacting string functions*

$$\begin{cases} f : a \to b / b\_\_ \\ g : b \to a / \_\_ a \end{cases} \qquad /bba/ \xrightarrow{g} b\,a\,a \xrightarrow{f} [b\,b\,a]$$

a. *$f \circ g$ violates condition (i) over the string 'bba'*

$/bba/ \xrightarrow{f} [bb\,b\,]$ $\quad \mu(f, bba) = \{2\}$

$/bba/ \xrightarrow{f \circ g} [bb\,a\,]$ $\quad [(f \circ g)(bba)]_2 \neq [f(bba)]_2$

b. *$f \circ g$ violates condition (ii) over the string 'bba'*

$/bba/ \xrightarrow{g} [b\,a\,a]$ $\quad \mu(g, bba) = \{1\}$

$/bba/ \xrightarrow{f \circ g} [b\,b\,a]$ $\quad [(f \circ g)(bba)]_1 \neq [g(bba)]_1$

[1] For example, condition (i) of the original definition simply said for all $x \in \mu(f)$, $(f \circ g)(x) = f(x)$. Since f and g are functions from strings to strings and x is an index of a particular string, the original definition needs some formal adjustments in order to capture the intended meaning.

(53) *Revisiting Example 5.2; $f \circ g$ violates condition (ii) over the string 'bcc'*

$$\begin{cases} f : a \to b / b\_\_ & /bcc/ \xrightarrow{g} [b\ a\ c] & \mu(g, bcc) = \{1\} \\ g : c \to a / \_\_c & /bcc/ \xrightarrow{g} b\ a\ c \xrightarrow{f} [b\ b\ c] & [(f \circ g)(bcc)]_1 \neq [g(bcc)]_1 \end{cases}$$

The conditions in (i) and (ii) are constraints against feeding and bleeding relationships between phonological maps. The two functions in (53), for example, demonstrate a *feeding* relationship over the string 'bcc'. The function f does not apply to 'bcc' because the environment for f is not found in 'bcc'. However, when g applies to 'bcc', it creates an environment for f to apply. It is for this reason that $(f \circ g)(bcc)$ yields a different output than $f(bcc)$. Condition (ii) of Definition 5.11 expresses a constraint against this type of relationship. Further discussion of feeding/bleeding relationships and process interaction in phonology can be found in Baković and Blumenfeld [2024], Baković [2011], McCarthy [2007].

5.2.1 Non-Interacting Composition of BMRS Programs

Definition 5.12 presents the concept of mutation points for BMRS programs. For any input string model, the mutation point of an output predicate P_σ^f is the set of indices in the input model for which P_σ^f flips the truth value associated with P_σ, as in (5.7). This definition extends to the mutation point of a program in the following way: x is a mutation point of a program F if and only if x is in a mutation point of some output predicate P_σ^f in F, as in (5.8). This captures the idea that the mutation points of F over an input model $\mathcal{M}$ are the collection of indices in $\mathcal{M}$ that undergo *some* change under the application of the program F.

Definition 5.12 (**Mutation Points: BMRS**). Let Σ be an input signature with predicates $\{P_\sigma\}_{\sigma \in \Sigma}$, and $F = \{P_\sigma^f, \vec{f}\}_{\sigma \in \Sigma}$ be program over Σ. For every model $\mathcal{M}$ with domain $D^{\mathcal{M}}$ over the signature Σ, the mutation points of each predicate P_σ^f over the input model $\mathcal{M}$ are defined as

$$\mu(P_\sigma^f, \mathcal{M}) := \{i \in D^{\mathcal{M}} \mid P_\sigma^f(i) \neq P_\sigma(i)\} \tag{5.7}$$

The mutation points of the *program* F over the input model $\mathcal{M}$ is defined as

$$\mu(F, \mathcal{M}) := \bigcup_{\sigma \in \Sigma} \mu(P_\sigma^f, \mathcal{M}) \tag{5.8}$$

The definition of mutation points for output predicates in (5.7) looks exactly the definition for mutation points for string functions in (5.6). The input model $\mathcal{M}$ in (5.7) is the BMRS representation of the input word w in (5.6). Given the intuitive equivalence between the concept of mutation points for string functions and for BMRS programs, we can give a BMRS-equivalent definition of non-interaction program composition in Definition 5.13. The BMRS-equivalent definition of a μ-conserving composition is immediately clear.

Definition 5.13 (**Non-Interacting Composition: BMRS**)**.** Let Σ be an input signature with predicates $\{P_\sigma\}_{\sigma\in\Sigma}$, and let $F = \{P^f_\sigma, \vec{f}\}_{\sigma\in\Sigma}$ and $G = \{P^g_\sigma, \vec{g}\}_{\sigma\in\Sigma}$ be programs over Σ. F and G do not interact when composed as $F \circ G$ iff for every model $\mathcal{M}$ over input signature Σ, the following two conditions holds:

(i) for all $i \in \mu(F, \mathcal{M})$, $P^{f\circ g}_\sigma(i) = P^f_\sigma(i)$ for all $\sigma \in \Sigma$

(ii) for all $i \in \mu(G, \mathcal{M})$, $P^{f\circ g}_\sigma(i) = P^g_\sigma(i)$ for all $\sigma \in \Sigma$

Condition (i) says that for every index i of the input model which undergoes change under the program F, the system of equations $\{P^{f\circ g}_\sigma\}_{\sigma\in\Sigma}$ will have the same solution at index i as the system of equations $\{P^f_\sigma\}_{\sigma\in\Sigma}$. In other words, the program $F \circ G$ will have the same solution at index i as the program F. This means that the program G does not make any changes that will block the application of F. In this way, condition (i) of Definition 5.13 is conceptually equivalent to condition (i) of Definition 5.11. Condition (ii) of the two definitions are equivalent in a similar manner. Thus, Definition 5.13 gives a BMRS interpretation of McCollum et al. [2018]'s definition of non-interacting function composition. An immediate consequence of this definition is that every non-interacting composition of two programs is μ-conserving. This is expressed in Proposition 5.14.

Proposition 5.14. If two programs F and G over the same input signature are non-interacting, then the composition $F \circ G$ is μ-conserving.

Proof. Let $F = \{P^f_\sigma\}_{\sigma\in\Sigma}$ and $G = \{P^g_\sigma\}_{\sigma\in\Sigma}$ be non-interacting programs over the signature Σ, and $\mathcal{M}$ be a model over Σ. To show that $F \circ G$ is μ-conserving, we show that (5.9) holds.

$$x \in \mu(F \circ G, \mathcal{M}) \quad \text{iff} \quad x \in \mu(F, \mathcal{M}) \cup \mu(G, \mathcal{M}) \tag{5.9}$$

($\Leftarrow$) Let $x \in \mu(F, \mathcal{M}) \cup \mu(G, \mathcal{M})$. Then there are two cases. Case 1: $x \in \mu(F, \mathcal{M})$. Then there is some $\sigma \in \Sigma$ such that $P^f_\sigma(x) \neq P_\sigma(x)$. By condition (i) of Def 5.13, $P^f_\sigma(x) = P^{f\circ g}_\sigma(x)$. Thus, $x \in \mu(F \circ G, \mathcal{M})$. Case 2: $x \in \mu(G, \mathcal{M})$. Then by similar reasoning and condition (ii), $x \in \mu(F \circ G, \mathcal{M})$. ($\Rightarrow$) Assume $x \notin \mu(F, \mathcal{M}) \cup \mu(G, \mathcal{M})$. Then $x \notin \mu(P^f_\sigma, \mathcal{M})$ and $x \notin \mu(P^g_\sigma, \mathcal{M})$ for any $\sigma \in \Sigma$. This means that $P^f_\sigma(x) = P_\sigma(x) = P^g_\sigma(x)$ for all σ. By definition of composition, $P^{f\circ g}_\sigma(x) = P_\sigma(x)$ for all σ and therefore, $x \notin \mu(F \circ G, \mathcal{M})$. □

Note that the converse of this proposition is *not* true. To see why this is the case, consider again the two functions $f : a \to b/b\_\_$ and $g : c \to a/\_\_c$ from (53). The BMRS programs for these functions, and their application over the string model $\mathcal{M}(bcc)$ were presented in Figure 5.3 in Example 5.2. Figure 5.11 presents the programs F, G, and $F \circ G$ over the model for 'bcc' in order to demonstrate why the converse does not hold. The three tables show that $\mu(F \circ G, \mathcal{M}(bcc)) = \mu(F, \mathcal{M}(bcc)) \cup \mu(G, \mathcal{M}(bcc))$. However, it was shown in (53) that $f \circ g$ (and consequently $F \circ G$) violate condition (ii) of the definition of non-interacting. Thus, being a μ-conserving composition does not guarantee that the composition is non-interacting.

	b	c	c
	0	1	2
$P_a(x)$	⊥	⊥	⊥
$P_b(x)$	⊤	⊥	⊥
$P_c(x)$	⊥	⊤	⊤
$P^f_a(x)$	⊥	⊥	⊥
$P^f_b(x)$	⊤	⊥	⊥
$P^f_c(x)$	⊥	⊤	⊤
	b	c	c

(a) $\mu(F, \mathcal{M}(bcc)) = \emptyset$

	b	c	c
	0	1	2
$P_a(x)$	⊥	⊥	⊥
$P_b(x)$	⊤	⊥	⊥
$P_c(x)$	⊥	⊤	⊤
$P^g_a(x)$	⊥	⊤	⊥
$P^g_b(x)$	⊤	⊥	⊥
$P^g_c(x)$	⊥	⊥	⊤
	b	a	c

(b) $\mu(G, \mathcal{M}(bcc)) = \{1\}$

	b	c	c
	0	1	2
$P^g_a(x)$	⊥	⊤	⊥
$P^g_b(x)$	⊤	⊥	⊥
$P^g_c(x)$	⊥	⊥	⊤
$P^{f\circ g}_a(x)$	⊥	⊥	⊥
$P^{f\circ g}_b(x)$	⊤	⊤	⊥
$P^{f\circ g}_c(x)$	⊥	⊥	⊤
	b	b	c

(c) $\mu(F \circ G, \mathcal{M}(bcc)) = \{1\}$

Figure 5.11: Demonstration of why the converse of Proposition 5.14 does not hold; $\mu(F \circ G, \mathcal{M}(bcc)) = \mu(F, \mathcal{M}(bcc)) \cup \mu(G, \mathcal{M}(bcc))$, but $F \circ G$ violates condition (ii) of Definition 5.13 of non-interacting compositions.

Proposition 5.14 however yields a meaningful connection between non-interacting compositions and simultaneous application. To formalize this relationship, we firsts generalize the concept of μ-conserving to any operator over BMRS programs in Definition 5.15. Proposition 5.16 then shows that simultaneous application of two programs is always μ-conserving. Thus, non-interacting composition and simultaneous application are both μ-conserving operations.

Definition 5.15 (**μ-Conserving Operators**)**.** Let $F = \{P_\sigma^f, \vec{f}\}_{\sigma\in\Sigma}$ and $G = \{P_\sigma^g, \vec{g}\}_{\sigma\in\Sigma}$ be programs over the input signature Σ. For any operator $\otimes$ over programs, $F \otimes G$ is μ-conserving iff for every input model $\mathcal{M}$,

$$\mu(F \otimes G, \mathcal{M}) = \mu(F, \mathcal{M}) \cup \mu(G, \mathcal{M})$$

Proposition 5.16. For any two programs F and G over the same input signature, the simultaneous application $F \circledcirc G$ is μ-conserving.

Proof. Let $F = \{P_\sigma^f\}_{\sigma\in\Sigma}$ and $G = \{P_\sigma^g\}_{\sigma\in\Sigma}$ be programs over the signature Σ, and $\mathcal{M}$ be a model over Σ. We show first that for any $\sigma \in \Sigma$, the 'iff' statement in (5.10) holds.

$$P_\sigma^{f\circledcirc g}(x) \neq P_\sigma(x) \text{ iff } P_\sigma^f(x) \neq P_\sigma(x) \text{ or } P_\sigma^g(x) \neq P_\sigma(x) \tag{5.10}$$

Case 1: $P_\sigma(x) = \top$. By (5.3) in Definition 5.4, $P_\sigma^{f\circledcirc g}(x)$ evaluates to $P_\sigma^f(x) \wedge P_\sigma^g(x)$. In this case, $P^{f\circledcirc g}(x) = \bot$ iff $P_\sigma^f(x) = \bot$ or $P_\sigma^f(x) = \bot$. Case 2: $P_\sigma(x) = \bot$. Then $P_\sigma^{f\circledcirc g}(x)$ evaluates to $P_\sigma^f(x) \vee P_\sigma^g(x)$. In this case, $P_\sigma^{f\circledcirc g}(x) = \top$ iff $P_\sigma^f(x) = \top$ or $P_\sigma^g(x) = \top$.

(5.10) immediately entails $\mu(P_\sigma^{f\circledcirc g}, \mathcal{M}) = \mu(P_\sigma^f, \mathcal{M}) \cup \mu(P_\sigma^g, \mathcal{M})$ for every $\sigma \in \Sigma$, and therefore

$$\mu(F \circledcirc G, \mathcal{M}) = \bigcup_{\sigma\in\Sigma} \mu(P_\sigma^{f\circledcirc g}, \mathcal{M}) = \bigcup_{\sigma\in\Sigma} \mu(P_\sigma^f, \mathcal{M}) \cup \mu(P_\sigma^g, \mathcal{M}) = \mu(F, \mathcal{M}) \cup \mu(G, \mathcal{M})$$

□

The main result of this section is expressed in Theorem 5.17. This result is the formal foundation for the discussion of learning non-interacting function compositions in the remainder of the chapter.

Theorem 5.17. If F and G are two programs over the same input signature and $F \circ G$ is non-interacting, then $F \circ G = F \circledcirc G$.

Proof. Let $F = \{P_\sigma^f\}_{\sigma\in\Sigma}$ and $G = \{P_\sigma^g\}_{\sigma\in\Sigma}$ be two programs over the same input signature Σ such that $F \circ G$ is non-interacting. Consider any model $\mathcal{M}$ over Σ with domain $D^{\mathcal{M}}$. It suffices to show that for every $\sigma \in \Sigma$ and every $x \in D^{\mathcal{M}}$

$$P_\sigma^{f\circ g}(x) = P_\sigma^{f\circledcirc g}(x)$$

Let $\sigma \in \Sigma$ and $x \in D^{\mathcal{M}}$. We consider three cases: Case 1: $x \in \mu(P^f_\sigma, \mathcal{M})$. Then $x \in \mu(P^{f \mathbin{⦶} g}_\sigma, \mathcal{M})$ because $⦶$ is μ-conserving (Proposition 5.16) and $P^{f \circ g}_\sigma(x) = P^f_\sigma(x)$ by condition (i) of the definition of non-interacting compositions (Definition 5.13). The facts together mean $P^{f \mathbin{⦶} g}_\sigma(x) \neq P_\sigma(x)$ and $P^{f \circ g}_\sigma(x) \neq P_\sigma(x)$. Since these are Boolean predicates, it follows that $P^{f \circ g}_\sigma(x) = P^{f \mathbin{⦶} g}_\sigma(x)$. Case 2: $x \in \mu(P^g_\sigma, \mathcal{M})$. Then $x \in \mu(P^{f \mathbin{⦶} g}_\sigma, \mathcal{M})$ because $⦶$ is μ-conserving, and $P^{f \circ g}_\sigma(x) = P^g_\sigma(x)$ by condition (ii) of non-interacting compositions. Thus by the same reasoning of Case 2, we get that $P^{f \circ g}_\sigma(x) = P^{f \mathbin{⦶} g}_\sigma(x)$. Case 3: $x \notin \mu(P^f_\sigma, \mathcal{M})$ and $x \notin \mu(P^g_\sigma, \mathcal{M})$. Then $P^f_\sigma(x) = P_\sigma(x) = P^g_\sigma(x)$, from which it follows that $P^{f \circ g}_\sigma(x) = P_\sigma(x) = P^{f \mathbin{⦶} g}_\sigma(x)$. □

5.2.2 Revisiting Canonical Normal Form

Recall from Definition 4.1 that for an input signature Σ with predicates $\{P_\sigma\}_{\sigma \in \Sigma}$, a non-recursive output predicate P'_σ is in canonical normal form (CNF) if and only if it can be expressed as (4.2), where $\phi_\sigma(x)$ and $\psi_\sigma(x)$ are either $\top$, $\bot$, or disjunctive normal form formulas over $atoms(\Sigma)$. For every $\sigma \in \Sigma$, ϕ_σ expresses the environment in which a σ input becomes a non-σ output, and ψ_σ expresses the environment in which a non-σ input becomes a σ output.

$$P'_\sigma(x) = \text{if } P_\sigma(x) \text{ then } \neg\phi_\sigma(x) \text{ else } \psi_\sigma(x) \tag{4.2}$$

Section 5.1 defined the simultaneous application of arbitrary programs. Here we consider the simultaneous application of CNF programs, and show that there is a very simple systematic process for computing the simultaneous application of two CNF expressions as an expression over $atoms(\Sigma)$. Consider again the two functions $f : a \to b/b\_\_$ and $g : a \to b/\_\_b$. The CNF of these programs are given in (54). The CNF of f was previously discussed in Section 4.1.1. The simultaneous application of these two programs is presented in (55).

(54) *Two non-interacting CNF BMRS programs*

a. $f : a \to b/b\_\_$ *as a CNF program*

$$\begin{aligned} P^f_a(x) &= \text{if } P_a(x) \text{ then } \neg P_b(p(x)) \text{ else } \bot \\ P^f_b(x) &= \text{if } P_b(x) \text{ then } \neg\bot \text{ else } \big(P_a(x) \wedge P_b(p(x))\big) \end{aligned}$$

b. $g : a \to b/\_\_b$ *as a CNF program*

$$
\begin{aligned}
P_a^g(x) &= \text{if } P_a(x) \text{ then } \neg P_b(s(x)) \text{ else } \bot \\
P_b^g(x) &= \text{if } P_b(x) \text{ then } \neg\bot \text{ else } \big(P_a(x) \wedge P_b(s(x))\big)
\end{aligned}
$$

(55) *Simultaneous Application of the CNF programs in (54)*

a. *Simultaneous application of P_a^f and P_a^g*

$$
\begin{aligned}
P_a^{f \circledcirc g}(x) &= \text{if } P_a(x) \text{ then } \big(P_a^f(x) \wedge P_a^g(x)\big) \text{ else } \big(P_a^f(x) \vee P_a^g(x)\big) \\
&\equiv \text{if } P_a(x) \text{ then } \big(\neg P_b(p(x)) \wedge \neg P_b(s(x))\big) \text{ else } \big(\bot \wedge \bot\big) \\
&\equiv \text{if } P_a(x) \text{ then } \neg \big(P_b(p(x)) \vee P_b(s(x))\big) \text{ else } \bot
\end{aligned}
$$

b. *Simultaneous application of P_b^f and P_b^g*

$$
\begin{aligned}
P_b^{f \circledcirc g}(x) &= \text{if } P_b(x) \text{ then } \big(P_b^f(x) \wedge P_b^g(x)\big) \text{ else } \big(P_b^f(x) \vee P_b^g(x)\big) \\
&\equiv \text{if } P_b(x) \text{ then } \big(\neg\bot \wedge \neg\bot\big) \text{ else } \big((P_a(x) \wedge P_b(p(x))) \vee (P_a(x) \wedge P_b(s(x)))\big) \\
&\equiv \text{if } P_b(x) \text{ then } \neg\bot \text{ else } \big((P_a(x) \wedge P_b(p(x))) \vee (P_a(x) \wedge P_b(s(x)))\big)
\end{aligned}
$$

Consider first the simultaneous application of P_a^f and P_a^g in (55a). The first line is the definition of the simultaneous application operator from Definition 5.4. When $P_a(x)$ evaluates to $\top$, $P_a^f(x)$ in (54a) evaluates to $\neg P_b(px)$, and $P_a^g(x)$ in (54b) evaluates to $\neg P_b(sx)$. The 'then' part of $P_a^{f \circledcirc g}(x)$ is therefore the conjunction of these two expression. Similarly, if $P_a(x)$ evaluates to $\bot$, $P_a^f(x)$ and $P_a^g(x)$ both evaluate to $\bot$. The 'else' part is therefore the disjunction of these two expressions. The final line of (55a) is then simplified to a CNF expression. The highlighted formula represents $\phi_a(x)$: an 'a' input becomes a non-'a' output when it is either preceded or followed by a 'b'. By similar reasoning, we get the CNF expression for $P_b^{f \circledcirc g}(x)$ in (55b). The highlighted formula represents $\psi_b(x)$: a non-'b' input becomes a 'b' output if it is either an 'a' that is preceded by a 'b', or it is an 'a' that is followed by a 'b'. It is immediately clear from the CNF representations that the function which simultaneous application models is (5.11), which says an 'a' becomes a 'b' if it is either preceded or followed by a 'b'.

$$
f \circledcirc g : a \to b/ \left\{ \begin{matrix} b\_ \\ \_b \end{matrix} \right\} \tag{5.11}
$$

Because the two programs in (54) are non-interacting, it follows from Theorem 5.17 that the program in (55) represents the *composition* of these two programs. It terms of string functions, it is intuitively clear that the function $f \circledcirc g$ in (5.11) is equivalent to the composition of f and g.

Moreover, the syntactic representation of this function as a BMRS program in (55) is very simple and intuitive. For comparison, consider the composition of P_a^f and P_a^g in (56). The first line of $P_a^{f\circ g}(x)$ is the definition of composition (Definition 5.1). The long expression in the second line is the result of substituting P_a^g and P_b^g with their definitions from (54b). Although the expression is defined entirely in terms of the input signature, it is hard to read and not clear how to simplify. More importantly, it is unclear how we might go about *learning* this complex expression for $P_a^{f\circ g}$. The fact that this complex expression can instead be represented as (55a) is therefore a significant advance toward the goal of learning composition.

(56) *Composition of P_a^f and P_a^g*

$$\begin{aligned} P_a^{f\circ g}(x) &= \text{if } P_a^g(x) \text{ then } \neg P_b^g(p(x)) \text{ else } \bot \\ &\equiv \text{if } \big(\text{if } P_a(x) \text{ then } \neg P_b(s(x)) \text{ else } \bot\big) \\ &\quad \text{then } \neg\big(\text{if } P_b(px) \text{ then } \neg\bot \text{ else } \big(P_a(px) \wedge P_b(x)\big)\big) \\ &\quad \text{else } \bot \end{aligned}$$

The example of simultaneous application in (54-55) can be generalized to any CNF programs. The general case for any two expressions is given in (57). It is clear that this can be generalized further to any number of expressions. Thus, the simultaneous application of any number of CNF programs can be expressed as a CNF program by simply taking the disjunction of all the ϕ_σ environments to form the 'then' part of the expression, and the disjunction of all the ψ_σ environments to form the 'else' part. The next section considers this syntactic process in reverse: given the expression in (57b), can we derive the two expressions in (57a)? For non-interacting processes, doing so would yield a syntactic way of *decomposing* a BMRS program.

(57) *Simultaneous Application of any two CNF expressions*

a.
$$\begin{aligned} P_\sigma^f(x) &= \text{if } P_\sigma(x) \text{ then } \neg\phi_\sigma^f(x) \text{ else } \psi_\sigma^f(x) \\ P_\sigma^g(x) &= \text{if } P_\sigma(x) \text{ then } \neg\phi_\sigma^g(x) \text{ else } \psi_\sigma^g(x) \end{aligned}$$

b. $P_\sigma^{f\circledcirc g}(x) = \text{if } P_\sigma(x) \text{ then } \neg(\phi_\sigma^f(x) \vee \phi_\sigma^g(x)) \text{ else } \big(\phi_\sigma^f(x) \vee \psi_\sigma^g(x)\big)$

5.3 Decomposition

This section reconciles the formal content of the previous two sections with the learning procedure introduced in Chapter 4. To summarize, Section 5.1 defined function composition and simultane-

ous application as operators over BMRS programs, and Section 5.2 showed that non-interacting compositions of programs can be expressed as simultaneous application. Section 5.2.2 in particular revisited the CNF used for learning in Chapter 4, to show how it is useful for efficiently computing the simultaneous application of two programs, and consequently, the composition of two non-interacting programs. This section discusses how CNF programs allow us to go in reverse as well; given a CNF BMRS program we can break it down into smaller programs. This observation is then discussed in the context of learning non-interacting compositions in Section 5.3.2.

5.3.1 Decomposition of Normal Form Programs

Section 5.2.2 presented a syntactic procedure for taking two CNF programs and expressing their simultaneous application as a CNF program. In this section, we reverse the procedure. That is, given a CNF program, we can determine the two programs that it is the simultaneous application of. Consider again the two non-interacting programs in (54), repeated below, and their simultaneous application in (58). The simultaneous application was previously derived in (55). The problem of decomposing this program into smaller programs is a syntactic one, which comes down to parsing the individual components $\{\phi_a^{f \circledcirc g}, \psi_a^{f \circledcirc g}, \phi_b^{f \circledcirc g}, \psi_b^{f \circledcirc g}\}$ of the program $\{P_a^{f \circledcirc g}, P_b^{f \circledcirc g}\}$.

(54) *Two non-interacting CNF BMRS programs*

a. $f : a \to b/b\_\_$ *as a CNF program*

$$P_a^f(x) = \text{if } P_a(x) \text{ then } \neg P_b(p(x)) \text{ else } \bot$$

$$P_b^f(x) = \text{if } P_b(x) \text{ then } \neg\bot \text{ else } \big(P_a(x) \wedge P_b(p(x))\big)$$

b. $g : a \to b/\_\_b$ *as a CNF program*

$$P_a^g(x) = \text{if } P_a(x) \text{ then } \neg P_b(s(x)) \text{ else } \bot$$

$$P_b^g(x) = \text{if } P_b(x) \text{ then } \neg\bot \text{ else } \big(P_a(x) \wedge P_b(s(x))\big)$$

(58) *Simultaneous Application of the CNF programs in (54)*

$$P_a^{f \circledcirc g}(x) = \text{if } P_a(x) \text{ then } \neg \big(P_b(p(x)) \vee P_b(s(x))\big) \text{ else } \bot$$

$$P_b^{f \circledcirc g}(x) = \text{if } P_b(x) \text{ then } \neg\bot \text{ else } \big((P_a(x) \wedge P_b(p(x))) \vee (P_a(x) \wedge P_b(s(x)))\big)$$

There are *seven* unique non-trivial syntactic decompositions of (58) into two smaller programs. These are presented in Figure 5.12, where each set of four rows represents two programs. The two rows above the dashed line represent $\{P_a^f(x), P_b^f(x)\}$ and the two rows below the dashed line represent $\{P_a^g(x), P_b^g(x)\}$. Row (7) of the table corresponds with the two programs in (54).

	'then'	'else'
(1)	$\neg(P_b(p(x)) \vee P_b(s(x)))$	$\bot$
	$\neg\bot$	$\bot$
	$\neg\bot$	$\bot$
	$\neg\bot$	$(P_a(x) \wedge P_b(p(x))) \vee (P_a(x) \wedge P_b(s(x)))$
(2)	$\neg P_b(p(x)) \vee P_b(s(x)))$	$\bot$
	$\neg\bot$	$P_a(x) \wedge P_b(p(x))$
	$\neg\bot$	$\bot$
	$\neg\bot$	$P_a(x) \wedge P_b(s(x))$
(3)	$\neg(P_b(p(x)) \vee P_b(s(x)))$	$\bot$
	$\neg\bot$	$P_a(x) \wedge P_b(s(x))$
	$\neg\bot$	$\bot$
	$\neg\bot$	$P_a(x) \wedge P_b(p(x))$
(4)	$\neg P_b(p(x))$	$\bot$
	$\neg\bot$	$\bot$
	$\neg P_b(s(x)))$	$\bot$
	$\neg\bot$	$(P_a(x) \wedge P_b(p(x))) \vee (P_a(x) \wedge P_b(s(x)))$
(5)	$\neg P_b(p(x))$	$\bot$
	$\neg\bot$	$(P_a(x) \wedge P_b(p(x))) \vee (P_a(x) \wedge P_b(s(x)))$
	$\neg P_b(s(x)))$	$\bot$
	$\neg\bot$	$\bot$
(6)	$\neg P_b(p(x))$	$\bot$
	$\neg\bot$	$P_a(x) \wedge P_b(s(x))$
	$\neg P_b(s(x)))$	$\bot$
	$\neg\bot$	$P_a(x) \wedge P_b(p(x))$
(7)	$\neg P_b(p(x))$	$\bot$
	$\neg\bot$	$P_a(x) \wedge P_b(p(x))$
	$\neg P_b(s(x)))$	$\bot$
	$\neg\bot$	$P_a(x) \wedge P_b(s(x))$

Figure 5.12: All unique and non-trivial decompositions of the program in (58) into two separate programs that it can be expressed as the simultaneous application of. The decomposition in (7) corresponds with the two programs in (54).

Although there are seven possible decompositions of (58), only the decomposition in (7) corresponds with two programs that represent valid string functions. Consider, for example, the two programs represented by the set of rows (1); the application of these two programs over the string model for 'aba' is presented in Figure 5.13. The program $\{P_a^f, P_b^f\}$ in (a) corresponds with the two rows above the dotted line of (1) of Figure 5.12, and the program $\{P_a^g, P_b^g\}$ in (b) corresponds with the two rows below the dotted line. The predicates P_a^f and P_b^f both evaluate to $\bot$ for the two 'a' inputs. The output model therefore doesn't yield any output for indices 0 and 2. The predicates P_a^g and P_b^g, on the other hand, both evaluate to $\top$ for the two 'a' inputs. The output model is

therefore uninterpretable for indices 0 and 2.

	a	b	a
	0	1	2
$P_a^f(x)$	$\bot$	$\bot$	$\bot$
$P_b^f(x)$	$\bot$	$\top$	$\bot$
		b	

(a)

	a	b	a
	0	1	2
$P_a^g(x)$	$\top$	$\bot$	$\top$
$P_b^g(x)$	$\top$	$\top$	$\top$
	!	b	!

(b)

Figure 5.13: The two programs represented by set of rows (1) of Figure 5.12 over the string model for 'aba'. Neither one models a well-defined string function.

To summarize, when programs are in normal form, there is a syntactic procedure for determining the simultaneous application of the two programs, which moreover yields a CNF program. We can use this procedure in the opposite direction: given a CNF program, we can decompose it to smaller programs that it is the simultaneous application of using the same syntactic procedure in reverse. The example of the two functions f and g from (54) illustrated the composition and decomposition processes. Even though there are several syntactically possible decompositions, the two programs in (54) are the *unique decomposition* of the program in (58). In other words, there is a unique solution to the question of whether the program in (54) can be decomposed into two programs F and G such that it is equivalent to $F \circ G$. The next section discusses this decomposition process with respect to feature models, and its implications for learning non-interacting compositions.

5.3.2 Decomposition and Learning

Recall that in a string model over an alphabet Σ, for every index x there must be exactly one $\sigma \in \Sigma$ such that $P_\sigma(x) = \top$. A BMRS program over string models is well-formed if for ever input string model, there is an output string model. This means that at every index x there is exactly one $\sigma \in \Sigma$ such that the *output* predicate $P'_\sigma(x)$ evaluates to $\top$. This property of string transductions yields programs which have interconnected parts. Consider for example a string function where an 'a' transforms into a 'b'. Then ϕ_a and ψ_b are interconnected; the environment which triggers an 'a' to become a non-'a' is the same environment which triggers a non-'b' to become a 'b'. These interconnected parts are the reason the program in (58) had seven possible non-trivial decompositions, six of which were not well-defined string transductions. Feature models do not have the requirement that exactly one predicate hold at each index. This means that the lines of a BMRS program for phonological maps over feature models do not have interconnected

pieces. This section shows how the problem of decomposing a BMRS program over feature models is significantly simpler for this reason. We consider two examples that illustrate this point.

The first example, presented in (59), involves two vowel alternation processes. The first process is vowel shortening (VS), in which a long vowel becomes short when it is followed by a consonant cluster. The second process is vowel lowering (VL), in which high long vowel become [-high]. When these two processes are ordered such that VL applies before VS, then the composition VS $\circ$VL is equivalent to the simultaneous application VS $\owedge$VL. For any word which has a high long vowel followed by two consonants, the high long vowel will undergo both lowering and shortening.

(59) *Two vowel alternation processes where VS$\circ$VL = VS$\owedge$VL*

VS: V: $\rightarrow$ V / __CC	*[Vowel Shortening]*
VL: V: $\rightarrow$ [-high]	*[Vowel Lowering]*

The two processes in (59) can be found in the Yawelmani dialect of Yokuts [Kuroda, 1967, Kenstowicz and Kisseberth, 1979]. Verb forms of the stem /mi:k/ 'swallow' in Yawelmani are presented in (60). The nonfuture form of the verb /mi:k-hin/ surfaces as [mek-hin], where the vowel /i:/ shortens and lowers to [e]. The CNF program which models input-output transformations such as /mi:k-hin/ $\rightarrow$ [mek-hin] is presented in (61) and Figure 5.14.

(60) *Verb forms for 'swallow' in Yawelmani [Kenstowicz and Kisseberth, 1977, pg.83]*

/mi:k/	Verb stem
mek'hin	Nonfuture
mek'k'a	Nmperative
me:k'al	Dubitative
me:ken	Future

(61) *Composition of vowel lowering and shortening (VS$\circ$VL) as a CNF BMRS program*

$$[long]'(x) = \text{if } [long](x) \text{ then } \neg(\, C(sx) \wedge C(ssx) \,) \text{ else } \bot$$

$$[high]'(x) = \text{if } [high](x) \text{ then } \neg\, [long](x) \text{ else } \bot$$

The significance of the representation in (61) is that it can be decomposed into two separate programs simply from its syntactic form and the reasoning discussed in the previous section. Moreover, there is a *unique* way to decompose the program, which is presented in (62). It is easy to check that the program in (61) is indeed the simultaneous application of the programs in (62a) and (62b). The program $\{[long]^f, [high]^f\}$ in (62a) is the CNF program for vowel shortening. Similarly, the program $\{[long]^g, [high]^g\}$ in (62b) is the CNF program for vowel lowering. In other words, the

	m	i:	k	h	i	n
$C(x)$			⊤	⊤		
$[long](x)$		⊤				
$[high](x)$		⊤				
$[long]'(x)$		⊥				
$[high]'(x)$		⊥				
	m	e	k	h	i	n

Figure 5.14: The program in (61) which models the composition of vowel lowering and shortening in Yawelmani, over the input /mi:khin/. Only relevant truth values are given to show how this program yields the output [mekhin].

CNF representation of VS ∘VL can be parsed into CNF representations and VS and VL from its syntactic form. Figure 5.15 presents these two programs over the input /mi:khin/.

(62) *Decomposition of the program in (61) into two programs*

a.
$$[long]^f(x) = \text{if } [long](x) \text{ then } \neg(\, C(sx) \wedge C(ssx)\,) \text{ else } \bot$$
$$[high]^f(x) = \text{if } [high](x) \text{ then } \neg\bot \text{ else } \bot$$

b.
$$[long]^g(x) = \text{if } [long](x) \text{ then } \neg\bot \text{ else } \bot$$
$$[high]^g(x) = \text{if } [high](x) \text{ then } \neg\, [long](x) \text{ else } \bot$$

	m	i:	k	h	i	n
$C(x)$			⊤	⊤		
$[long](x)$		⊤				
$[high](x)$		⊤				
$[long]^f(x)$		⊥				
$[high]^f(x)$		⊤				
	m	i	k	h	i	n

(a)

	m	i:	k	h	i	n
$C(x)$			⊤	⊤		
$[long](x)$		⊤				
$[high](x)$		⊤				
$[long]^g(x)$		⊤				
$[high]^g(x)$		⊥				
	m	e:	k	h	i	n

(b)

Figure 5.15: The individual programs in (62) over the input model /mi:khin/. The table in (a) represents the program in (62a), which models vowel shortening. The table in (b) represents the program (62b), which models vowel lowering.

From the perspective of learning, this demonstration shows that it is possible to learn the two individual processes from a sample that contains input-output pairs from their *composition* when the composition is equivalent to the simultaneous application of the two processes. If the learning procedure in Chapter 4 were given the input-output pair of models $(mi{:}khin, mekhin)$, it would update two hypothesis spaces: $\mathcal{H}(\psi_{[long]})$ and $\mathcal{H}(\psi_{[high]})$ because the second sound in the input model has an index where a [+long] sound becomes [-long], and a [+high] sound becomes [-high].

As the learning procedure updates $\mathcal{H}(\psi_{[long]})$ and $\mathcal{H}(\psi_{[high]})$, the corresponding BMRS program can be decomposed into two separate programs, one which represents a hypothesis for the shortening process, and one which represents a hypothesis for the lowering process. In this way, the learner is learning the two separate processes simultaneously from their composition. As each of the spaces $\mathcal{H}(\psi_{[long]})$ and $\mathcal{H}(\psi_{[high]})$ get updated, so does the program that represents the composition VL $\circ$VS. Since this composition is equivalent to VL $\owedge$VS, it can be decomposed into the individual programs representing VL and VS. If the learner ultimately yields the program in (61), it is then possible to break this program down to the two programs in (62).

In the vowel shortening and lowering example, two separate features undergo change. The second example, presented in (63), involves two voicing alternation processes. The first process is final devoicing (FDV), in which a voiced obstruent at the end of a word becomes voiceless. The second process is regressive voice assimilation (RVA), in which a voiceless obstruents becomes voiced when they are immediately followed by a voiced stop. Similar to the vowel alternations in the previous example, FDV $\circ$ RVA is equivalent to the simultaneous application FDV $\owedge$ RVA.

(63) *Two voicing alternation processes where FDV$\circ$RVA = FDV$\owedge$RVA*

FDV: [-son] $\rightarrow$ [-voi] / __# *[Word-Final Devoicing]*

RVA: [-son] $\rightarrow$ [+voi] / __[-son,-cont,+voi] *[Regressive Voice Assimilation]*

The CNF BMRS program for the composition/simultaneous application of final devoicing and regressive assimilation is presented in (64). Since the only feature that undergoes change is [voi], (64) only specifies the output predicate [voi]$'$. The highlighted expressions correspond with $\phi_{[voi]}$ and $\psi_{[voi]}$. $\phi_{[voi]}(x) = \texttt{final}(x)$ represents the fact that a [+voi] input becomes [-voi] if it is the final index in a word. Similarly, $\psi_{[voi]}(x) = \neg[son](sx) \wedge \neg[cont](sx) \wedge [voi](sx)$ represents the fact that a [-voi] input becomes [+voi] when it is followed by a voiced stop. Both processes can be found in Dutch [Booij, 1999].

(64) *Composition of final devoicing and regressive assimilation as a CNF BMRS program*

$[voi]'(x) =$ if $[voi](x)$ then $\neg\, \texttt{final}(x)$ else $\neg[son](sx) \wedge \neg[cont](sx) \wedge [voi](sx)$

Similar to the vowel alternation processes in the previous example, there is a unique way to decompose (64) into two separate programs. These are presented in (65). The expression in (65a) corresponds with word-final devoicing, and the expression in (65b) corresponds with regressive voice

assimilation. It is easy to see that $[voi]^f(x) \veebar [voi]^g(x) = [voi]'(x)$. As with the previous example, if the learning procedure in Chapter 4 sees input-output pairs form the composition FDV $\circ$ RVA, it will update two hypothesis spaces: $\mathcal{H}(\phi_{[voi]})$ and $\mathcal{H}(\psi_{[voi]})$. As each of these spaces are updated, the corresponding BMRS program can be decomposed into two separate programs, one representing a hypothesis for the devoicing processes and one representing a hypothesis for the voicing process. Thus, voicing and devoicing are learned simultaneously from their composition.

(65) *Decomposition of the program in (64) into two programs*

a. $[voi]^f(x) =$ if $[voi](x)$ then $\neg$ `final`(x) else $\bot$

b. $[voi]^g(x) =$ if $[voi](x)$ then $\neg\bot$ else $\neg[son](sx) \wedge \neg[cont](sx) \wedge [voi](sx)$

The two examples presented here ultimately show that when when a composition of two processes is equivalent to their simultaneous application, the learning procedure described in Chapter 4 amounts to learning each of the individual processes simultaneously. This is because the output of the learning procedure is a CNF program, which can then be decomposed into CNF programs that it is the simultaneous application of. This is particularly significant given Theorem 5.17 which states that if two programs are non-interacting, then their composition is equivalent to simultaneous application. In terms of learning, this means that the work in Chapter 4 can be extended to learning non-interacting compositions.

5.4 Discussion

The first part of this chapter defined composition and simultaneous application as operators over BMRS programs. With these definitions, it was possible to formalize concepts like 'non-interacting composition' in terms of BMRS programs. The main contribution of the first half of this chapter was Theorem 5.17 which states that whenever two programs are non-interacting, their composition is equivalent to simultaneous application. The second half of the chapter discussed this result within the context of learning compositions. If F and G are non-interacting when composed as $F \circ G$, then $F \circ G = F \veebar G$ means that in order to decompose $F \circ G$ into the individual programs F and G, it is sufficient to decompose $F \veebar G$ into F and G. Section 5.3 showed that it is indeed possible to decompose a CNF BMRS program P into two separate programs F and G such that $P = F \veebar G$. In the case of feature models, in particular, there is a unique decomposition that is

easy to derive from the syntactic structure of CNF programs. The consequence of these results can therefore be stated as follows. Whenever two phonological maps f and g are non-interacting when composed as $f \circ g$, the learning procedure in Chapter 4.2 can be extended to a procedure for learning individual processes from their composition, in the special case where those processes or non-interacting.

An interesting challenge that this chapter invites is determining what constitutes an *individual* process. Consider again the two interacting functions $f : a \to b/b\_\_$ and $g : b \to a/\_\_a$ over the alphabet $\Sigma = \{a, b\}$ from (52) of Section 5.2. The $a \to b$ change made by the outer function f is observed in $f \circ g$ if the inner function g does not block it. In other words, $f \circ g : a \to b$ if the environment for f is present but the environment for g is not. Thus, $a \to b/b\_\_\neg a$. Similarly, the $b \to a$ change made by the inner function g is observed in $f \circ g$ if the outer function f does not undo the change. This means $b \to a/\neg b\_\_a$. The *composition* $f \circ g$ is the same string function as the *simultaneous application* of these two maps. This is explicitly written out in (5.12). thus, $f \circ g$ can be decomposed into two non-interacting processes that it is the composition of. This ambiguity adds an interesting challenge to the goal of learning individual processes from their composition.

$$(a \to b/b\_\_) \circ (b \to a/\_\_a) \quad = \quad (a \to b/b\_\_\neg a) \circledcirc (b \to a/\neg b\_\_a) \tag{5.12}$$

As a final note, one of the limitations of this chapter is that it is based off of Chapter 4, which only addresses input strictly local maps. The definition of CNF programs (Definition 4.1) is given specifically as a normal form for non-recursive and length-preserving transductions. While the definitions of composition, simultaneous application, and non-interacting composition in this chapter are not limited to non-recursive programs, the decomposition procedure is based on parsing CNF programs. The discussion of learning individual processes from their composition is therefore limited to local processes.

6

LOGICAL CHARACTERIZATION OF WEAKLY DETERMINISTIC MAPS

The Weakly Deterministic Hypothesis posits that phonological maps have a bias for weak determinism. The weakly deterministic class of functions carve out a space between subsequential and rational [Lamont et al., 2019, Heinz and Lai, 2013]. This chapter proposes a *logical* characterization of the weakly deterministic functions as a syntactic constraint on BMRS programs, and shows that such a characterization correctly identifies cross-linguistic patterns as weakly deterministic, and can be used to identify which patterns are *not* weakly deterministic. The consequence of this result is a formal characterization of the boundary between weakly deterministic and rational, and a *testable hypothesis* about the complexity of natural language phonological patterns. Section 6.1 presents cross-linguistics patterns which provide empirical motivation for the weakly deterministic class. Section 6.2 proposes a BMRS definition of weakly deterministic functions in terms of simultaneous application operator on BMRS programs discussed in Chapter 5, and shows that the proposed definition successfully models weakly deterministic functions *and* excludes unbounded circumambient ones. Section 6.3 compares the definition proposed here with previous proposals which have failed to correctly separate the weakly deterministic and rational (unbounded circumambient) maps.

6.1 The Weakly Deterministic Class of Functions

This section provides empirical motivations for the weakly deterministic class of functions, and presents the *informal* property that distinguishes weakly deterministic maps from rational (unbounded circumambient) maps. Section 6.1.1 presents several examples of phonological maps which lead to an intuitive concept of weak determinism that is shared by phonologists [Heinz and Lai, 2013, Meinhardt et al., 2024, 2021, McCollum et al., 2018]. Section 6.1.2 briefly discusses some of the characterizations of weak determinism proposed in previous literature in order to highlight the informal concept and the challenge of meeting this concept with a formal characterization.

6.1.1 Empirical Motivations

Section 3.1.2 discussed phonological maps which challenge the Subsequential Hypothesis, and presented Akan stem-controlled ATR harmony in (18), and repeated below, as an example of a weakly deterministic map and Yidiny liquid dissimilation in (19) as an example of an unbounded circumambient map. These examples demonstrated that there are two types of non-subsequential maps that are attested in natural language phonology. The purpose of this section is to highlight the difference between these two types of maps through further cross-linguistic examples, and identify the characteristic property that distinguishes the maps we classify as 'weakly deterministic'.

(18) *Stem-Controlled ATR Harmony in Akan [Clements, 1985]*

a. *[-ATR] Root*

ɔ-tsɪ̱rɛ̱-**ɪ**	'3S-show-3S.OBJ'
ɔ-b**ɛ**-tʊ̱-**ɪ**	'3S-FUT-throw-3S.OBJ'

b. *[+ATR] Root*

o-fi̱ti̱-**i**	'3S-pierce-3S.OBJ'
o-b**e**-tu̱-**i**	'3S-FUT-dig-3S.OBJ

Stem-controlled harmony refers to phonological maps in which the affixes of a word get their harmonic feature value from the the stem of affixation [Baković, 2000]. In languages where words are constructed by stacking suffixes onto the stem, harmony takes place left-to-right (i.e. left subsequential). Similarly, in languages where words are constructed by stacking prefixes, harmony is right subsequential. In languages like Akan, where prefixes and suffixes both stack onto the stem, stem-controlled harmony is *bidirectional*. As discussed in Section 3.1.2, this map is not subsequential

because the [ATR] value of prefixes rely on information found to the right while the [ATR] value of suffixes rely on information to the left. This property makes bidirectional stem-controlled harmony in Akan *weakly deterministic*.

Bidirectional harmony is also found outside of stem-controlled processes. Emphasis spread (pharyngeal harmony) in Palestinian Arabic [Davis, 1995, Watson, 1999, McCarthy, 1997] is a bidirectional harmony process that further exemplifies weakly deterministic maps [Jardine and Oakden, 2023]. Words with the emphatic phonemes /Ṭ/ and /Ṣ/ are presented in (66); the underlying forms of these words are provided in order to make the map clearer. Following Davis [1995], pharyngeal segments in these examples are capitalized. Leftward harmony is presented in (a); all the sounds in these words are pharyngealized because there is an emphatic /Ṭ/ or /Ṣ/ at the right edge of the word. The same harmony process also takes place in the opposite direction, with the exception that certain sounds can block harmony. In (b), all sounds are pharyngealized because there is a /Ṭ/ or /Ṣ/ at the *left* edge of the word. In (c), however, rightward harmony is blocked by high front vowels /i/ and /y/. The blocking of harmony by /i/ is observable in the contrast between [ṬUUB-AK] 'your blocks' and [Ṭiin-ak] 'your mud'. The second person possessive pronoun surfaces as [-AK] in the first word because there is an emphatic sound /Ṭ/ to the left, but surfaces as [-ak] in the second word because pharyngeal harmony is blocked by /i/. The final data point in (d) has the emphatic phoneme in the middle of the word to illustrate the bidirectional nature of this map; harmony takes place both leftward and rightward in a single word.

(66) *Pharyngeal Harmony in Palestinian Arabic* [Davis, 1995]

a. *Leftward harmony*

/xayyaaṬ/	XAYYAAṬ	'tailor'
/ballaaṢ/	BALLAAṢ	'thief'

b. *Rightward harmony without a blocker*

/Ṭuub-ak/	ṬUUB-AK	'your blocks'
/Ṣeef-ak/	ṢEEF-AK	'your sword'

c. *Rightward harmony with a blocker*

/Ṭiin-ak/	Ṭiin-ak	'your mud'
/Ṣayyad/	ṢAyyad	'hunter'

d. *Bidirectional harmony without a blocker*

/ʔaṬfall/	ʔAṬFALL	'children'

Every sound that undergoes pharyngeal harmony is the result of applying either a left or a

right subsequential function. In (a), sounds are pharyngealized because of an emphatic sound to the *right*; these input-output pairs can be modeled with a right subsequential function. In (b) and (c), whether a sound is pharyngealized depends on whether there is an emphatic phoneme and *no blocker* to the left; these input-output pairs can be modeled with a left subsequential function. The word in (d) shows how both processes happen together. Because the emphatic phoneme is in the middle of the word, every sound becomes pharyngealized either because of information on the left *or* information on the right. In this way, this map is very similar to Akan stem-controlled harmony.

Default-to-same stress processes [Hayes, 1995] are also weakly deterministic [Koser and Jardine, 2020, Hao and Andersson, 2019]. Leftmost-heavy-otherwise-leftmost (LHOL) stress is one kind of default-to-same stress process in which stress goes on the leftmost heavy syllable of a word and defaults to stressing the leftmost light syllable in a word with no heavy syllables. LHOL stress is found in Lushootseed [Hess, 1976, Koser and Jardine, 2020], Murik [Abbott, 1985, Walker, 1996], the Lhasa variety of Tibetan [Dawson, 1980, Odden, 1979, Gordon, 2006], and Yana [Sapir and Swadesh, 1960], among others. An example of LHOL is presented in (67) with words from Murik. H represents a syllable with a long vowel (i.e. heavy syllable) and L represents a syllable with a short vowel (i.e. light syllable). The words in (a) show that when a word has long vowels, the leftmost syllable with a long vowel gets stressed. The words in (b) show that when a word does not have any long vowels, the leftmost syllable gets default stress.

(67) LHOL Stress in Murik [Abbott, 1985]

a. *Stress leftmost heavy syllable*

anəp^{h}arɛːt^h	LLLH́	‘lightning’
numaɾóːgoː	LLH́H	‘woman’

b. *Default to leftmost syllable if no heavy syllables*

dámag	Ĺ L	‘garden’
dák^{h}anɨmp	Ĺ LL	‘post’

LHOL stress can be modeled with two separate subsequential processes: one which stresses the leftmost heavy syllable if it exists, and one which stresses the leftmost syllable if a heavy one does not exist. The first process requires information an unbounded direction to the left; a syllable is stressed if and only if it is heavy and there are no other heavy syllables anywhere *to the left* of it. The second process requires information an unbounded direction to the right; a syllable is stressed if and only if it is an initial light syllable and there are no heavy syllables

anywhere *to the right* of it. However, in every word, only one of these processes determines which syllable will get stressed. LHOL stress is therefore exactly like Akan stem-controlled harmony and Arabic pharyngeal harmony with respect to the kind of information needed to determine the surface form of a word. In all three examples, the overall function can be expressed as contradirectional subsequential functions such that every input-output relation between individual segments is the result of only one of these functions. LHOL stress shows that this property of phonological maps exists outside of the domain of vowel and consonant harmony.

The three weakly deterministic maps from Akan (18), Palestinian Arabic (66), and Murik (67) are notably different from the Sour Grapes harmony pathology [Padgett, 1995, Wilson, 2003], which is also outside of the subsequential class [Heinz and Lai, 2013]. Sour Grapes is presented in (68), where + represents a feature that is spreading, − represents a sound lacking the feature, and ⊟ represents a sound that blocks harmony. Harmony takes place if there is a trigger for harmony earlier in the string, and nothing blocking harmony later in the string. The characteristic property of Sour Grapes is that spreading of a feature occurs only if it can spread to the end of the word. In 68(a) the feature spreads to the end of the string while in (b) the feature does not spread at all because of the presence of a blocker ⊟ at the end of the word. Sour Grapes is an unbounded circumambient map; a '−' input becomes '+' if and only if there is a '+' anywhere to the left and there isn't a '⊟' anywhere to the right. In other words, this process requires non-local information in *both* directions. This property is what distinguishes it from the weakly deterministic ones.

(68) Sour Grapes Harmony [Padgett, 1995, Wilson, 2003]

a. $+ - - - - - \mapsto + + + + +$ (*Trigger for harmony, and no blocker*)

b. $+ - - - \boxminus \mapsto + - - - \boxminus$ (*Trigger for harmony, and a blocker*)

c. $- - - - - - \mapsto - - - - -$ (*No trigger for harmony*)

Sour Grapes was originally introduced as a 'pathological' map because it was believed to be unattested in natural language. These types of functions are, however, attested in tonal processes [Jardine, 2016a]. One such process is high tone spreading in Copperbelt Bemba [Bickmore and Kula, 2013, Kula and Bickmore, 2015], presented in (69). In Copperbelt Bemba, a high tone (◌́) spreads to the end of the word, unless there is another high tone anywhere to the right. The data points in (a) show how the high tone in /bá-/ spreads to all the vowels in a word. When there is another high tone anywhere to the right, this unbounded spreading is blocked and only the next two

vowels surface with a high tone; the remaining vowels surface with a low tone (◌̀). The blocking process is shown in the data points in (b), where /kó/ blocks the spreading of the high tone to the end of the word. In (c), there are no high tones in the underlying forms and so all vowels surface with low tones by default. High tone spreading in Copperbelt Bemba is unbounded circumambient [Jardine, 2016a]; whether or not any vowel surfaces with a high tone depends on the presence of a high tone an unbounded distance to the left and a blocking high tone an unbounded distance to the right.

(69) High Tone Spreading in Copperbelt Bemba [Bickmore and Kula, 2013]

a. *Unbounded high tone spreading*

/bá-ka-fik-a/ → [bá-ká-fík-á]	'they will arrive'
/bá-ka-mu-londolol-a/ → [bá-ká-mú-lóndólól-á]	'they will introduce him/her'

b. *Bounded high tone spreading*

/bá-ka-pat-a kó/ → [bá-ká-pát-à kó]	'they will hate'
/bá-ka-londolol-a kó/ → [bá-ká-ló-òndòlòl-à kó]	'they will introduce them'

c. *no high tones*

/u-ku-tul-a/ → [ù-kù-tùl-à]	'to pierce'

Although rare, McCollum et al. [2020] argues that unbounded circumambient maps can also be found in vowel harmony. One such example is ATR harmony in Tutrugbu [McCollum et al., 2020, McCollum and Essegbey, 2018, Essegbey, 2019, 2009], presented in (70). Prefixes in Tutrugbu get their ATR value from the root. The data points in (a) and (b) share the same set of prefixes but vary in the root, to illustrate the harmony process. In (a) the root /ʃē/ 'grow' has a [+ATR] vowel, and all the vowels in the prefixes surface as [+ATR]. For example, the first person plural (1P) prefix surfaces as [bu] and the third person singular (3S) prefix surfaces as [e]. In (b) the root /bá/ 'come' has a [-ATR] vowel, and all the prefixes surface as [-ATR]. In this case, 1P is pronounced as [bʊ] and 3S is pronounced as [a]. ATR harmony in Tutrugbu has *conditional blocking*; harmony is blocked by [-high] vowels, but only when the initial prefix is a [+high] vowel. This blocking process is presented in (c) with the [+ATR] root /wu/ 'climb'. The contrast between [i-tí-wu] and [ɪ-ba-wu] shows how the negation prefix /ba/ blocks harmony; in the former the 1S prefix becomes [+ATR] while in the latter it does not. On the other hand, the contrast between [ɪ-ba-wu] and [a-tí-ba-bá] shows that the negation prefix does not always block harmony; harmony is blocked in the former but not the latter because of the [+high] initial prefix in [ɪ-ba-wu]. This is illustrated in the final

two data points in (c) as well, where the only difference between the two words is whether the initial prefix is [+high].

(70) ATR Harmony in Tutrugbu [McCollum and Essegbey, 2018]

a. [+ATR] root

bu-tí-ʃē	'1P-NEG-grow'
e-tí-be-ʃē	'3S-NEG-FUT-grow'

b. [-ATR] root

bʊ-tí-bá	'1P-NEG-come'
a-tí-ba-bá	'3S-NEG-FUT-come'

c. *Conditional Blocking*

i-tí-wu	'1S-NEG-climb'
ɪ-ba-wu	'1S-FUT-climb'
ɪ-tí-ka-a-ba-ba-wu	'1S-NEG-PFV-PROG-VENT-VENT-climb'
e-tí-ke-e-be-be-wu	'3S-NEG-PFV-PROG-VENT-VENT-climb'

The ATR value of vowels in the prefixes therefore depends on three pieces of non-local information: the ATR value of the root, whether there is a [-high] vowel anywhere to the right, and whether the initial prefix is [+high]. That means the surface form of every vowel in the prefixes depends on information an unbounded distance away to its left and to its right, at the same time. This observation is what distinguishes it from stem-controlled ATR harmony in Akan, where prefix vowel only need information to the right and suffix vowels only need information to the left.

Section 3.1.2 presented Yidiny liquid dissimilation in (19) as an example of an unbounded circumambient map. Yidiny is particularly interesting because it is the only case of a non-subsequential dissimilation map in Payne [2017]'s survey of dissimilation processes found in **?**. In Yidiny, the 'going' affixes /ːli/ and /ŋali/ undergo dissimilation if there is another /l/ to the right. The dissimilation process gets blocked, however, if there is an /r/ in the stem to the left. This map is therefore similar to Sour Grapes; whether or not dissimilation takes place depends on the presence of a trigger /l/ to the right and a blocker /r/ to the left. Thus, while rare, consonant dissimilation can also be unbounded circumambient.

(19) *Liquid Dissimilation in Yidiny [Dixon, 1977]*

a. *Dissimilation*

magi-ːli-ɲu	'went climbing up'
magi-ːri-ŋaːl̪	'went climbing with'

duŋa-ŋali-ɲu	‘went running’
duŋ-ŋa**r**i-ŋaːl̲	‘went running with’

b. *Dissimilation blocking*

bur̲wa-ːli-ŋaːl̲	‘went jumping with’
bur̲g-ːli-ŋaːl̲	‘went walkbout with’

Figure 6.1 summarizes the collection of non-subsequential maps presented in this section. The goal of this paper is to present a formal characterization of weakly deterministic functions which realizes this figure. Previous characterizations have either incorrectly identified unbounded circumambient maps as weakly deterministic, or have not yet been proven successful in excluding the unbounded circumambient maps. A brief discussion of previous characterizations is discussed in the next subsection. Further discussion of the empirical difference between these two classes of functions can be found in Meinhardt et al. [2021, 2024] with a discussion of ATR harmony in Maasai and Turkana.

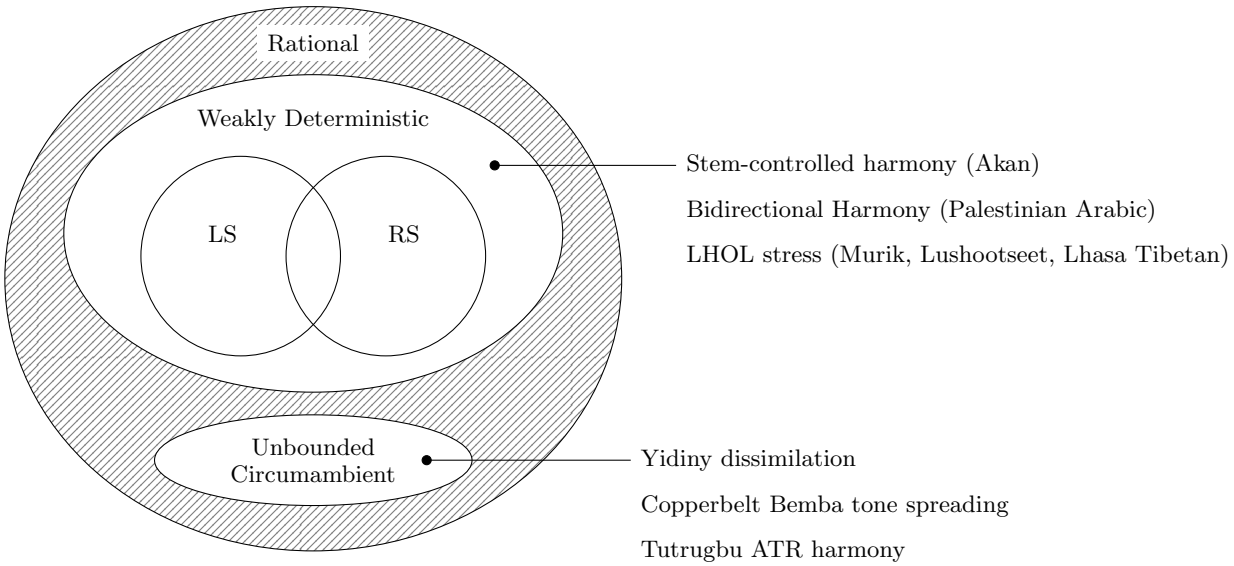


Figure 6.1: Classifications of the non-subsequential phonological maps discussed in this section into the weakly deterministic and unbounded circumambient classes.

6.1.2 Toward a Formal Characterization

Every rational function can be expressed as the composition of two contradirectional subsequential functions [Elgot and Mezei, 1965]. Because weakly deterministic functions are a proper subset of the regular functions [Lamont et al., 2019, Heinz and Lai, 2013], previous definition proposals have characterized the weakly deterministic functions with some restriction on function composition. Although it is possible to give an intuitive and informal description of what kinds of phonological processes ought to be considered weakly deterministic, providing a formal definition that includes *all and only* these maps has been an ongoing topic of research in computational phonology [Meinhardt et al., 2021, 2024, Heinz and Lai, 2013, McCollum et al., 2018]. A significant challenge to this goal is defining the class in such a way that it is possible to show when a map is *not* weakly deterministic. McCollum et al. [2018] and Lamont [2019] show that the definition proposed by Heinz and Lai [2013] is not successful in separating the weakly deterministic and unbounded circumambient maps, and definitions proposed by Meinhardt et al. [2021, 2024] have not been successfully used to show when a map is not weakly deterministic.

Heinz and Lai [2013] define the weakly deterministic functions as those which can be expressed as the composition of left and right subsequential functions *without the use of mark-up* on the input string. Their proposal is motivated by the fact that Elgot and Mezei use mark-up in order to decompose rational functions into the composition of two contradirectional functions. Consider again Sour Grapes harmony from (68). This function can be described as the composition of a left subsequential (LS) function which marks '−' inputs with a special symbol if there is a trigger for harmony on the left, and a right subsequential function (RS) which changes the special symbol to '+' if there isn't a blocker on the right, and back to '−' otherwise. This composition is presented in Figure 6.2. The special symbol Ψ allows the left subsequential function to give the right subsequential function information about the left side of the string through intermediate mark-up. Heinz & Lai conjecture that Sour Grapes harmony cannot be expressed as the composition of left and right subsequential functions without the use of such mark-up.

Mark-up as a restriction on function composition ultimately fails to separate weakly deterministic and unbounded circumambient patterns. Yidiny liquid dissimilation can be expressed as the composition of two subsequential dissimilation functions (double dissimilation) where the second

input	+	−	−	−
LS	+	Ψ	Ψ	Ψ
RS	+	+	+	+

(a) When the righthand side of the string does not have a blocker, RS rewrites Ψ as + because harmony is not blocked.

input	+	−	−	⊟
LS	+	Ψ	Ψ	⊟
RS	+	−	−	⊟

(b) When the righthand side of the string does have a blocker, RS rewrites Ψ as − because harmony is blocked.

Figure 6.2: Sour Grapes as the composition of a left subsequential (LS) and right subsequential (RS) function, using mark-up. The symbol Ψ encodes the information: there is a trigger for harmony somewhere on the left.

function undoes changes made by the first function without introducing new symbols [Dixon, 1977, Payne, 2017]. McCollum et al. [2018] shows that Tutrugbu ATR harmony and Copperbelt Bemba high tone spreading can also be expressed as a composition of two subsequential functions which do not introduce new symbols, by exploiting the surface phonotactics of the language. Using this same method, Lamont [2019] further shows that even Sour Grapes can be expressed as a composition of contradirectional subsequential functions without mark-up, thus disproving the conjecture. These are discussed in more detail in Section 6.3.

In response to Heinz & Lai's definition, Meinhardt et al. [2021] proposed a definition that relies on the concepts of mutation points and maps. Recall from Section 5.2 that the mutation points of a function f is the set of points $\{x \in dom(f) \mid f(x) \neq x\}$. This concept can be extended to the concept of a mutation *map*, which is the restriction of f to its mutation points, as in (6.1). A formal discussion of mutation maps for *string* functions is presented in Section 6.3.

$$\overrightarrow{\mu}(f) := \{(x, f(x)) \mid x \in \mu(f)\} \tag{6.1}$$

The motivation for using mutation maps is to be able to talk about the changes made by a left subsequential function and a right subsequential function independently. This idea is based on the observation that weakly deterministic functions are such that input-output relations between *individual segments* can be described in terms of a single subsequential function. This idea is illustrated in Figure 6.3 using the data point from Palestinian Arabic in (66d). This function can be expressed as the composition of a right subsequential (RS) function which spreads the pharyngeal feature leftward and a left subsequential (LS) function which spreads it rightward. The mutation map of the overall bidirectional map is the union of the mutation maps of LS and RS; this what

Meinhardt et al. [2021] identify as the distinguishing property of weakly deterministic compositions.

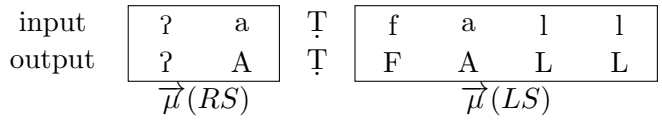


Figure 6.3: Mutation map for bidirectional harmony in Palestinian Arabic, over the input-output pair in (66d). The mutation map can be partitioned into two disjoint mutation maps; RS is leftward harmony and LS is rightward harmony.

Ultimately, the intuitions expressed by Heinz and Lai [2013] and Meinhardt et al. [2021] are the same. However, a significant problem for previous proposals is that they do not make it possible to prove when a function is *not weakly deterministic*. Because they characterize weak determinism in terms of a restriction on function composition, proving that something isn't weakly deterministic then requires showing that such a composition does not exist. While the definition proposed in this chapter captures all the same ideas and intuitions as Heinz and Lai [2013] and Meinhardt et al. [2021], it is situated in a framework that makes it possible to define weak determinism without making reference to function composition. Consequently, this makes it possible to prove when something is not weakly deterministic. As a result, this chapter provides the first definition of weakly deterministic functions that is successfully used to prove that Sour Grapes and other unbounded circumambient maps are not weakly deterministic.

6.2 Characterization of Weak Determinism

This chapter proposes a characterization of the weakly deterministic maps as those that can be expressed as the simultaneous application of a left and right subsequential function, as in Definition 6.1. This idea was previously proposed by Meinhardt et al. [2021]. However, this chapter formalizes this idea within the BMRS framework by using the simultaneous application operator from Definition 5.4 of Chapter 5.

Definition 6.1 (Weakly Deterministic Functions)**.** A string function f is weakly deterministic if and only if there exist contradirectional subsequential functions L and R such that f is the simultaneous application of L and R.

Recall from Section 3.2.2 that the class of BMRS programs which are defined over an input

signature that only has the predecessor function (BMRS^p) is the logical characterization of the left subsequential functions. Similarly, BMRS^s is the logical characterization of the right subsequential functions. The BMRS implementation of Definition 6.1 is therefore Definition 6.2.

Definition 6.2 (Weakly Deterministic Functions, BMRS)**.** A string function $f : \Sigma^* \to \Sigma^*$ is weakly deterministic if and only if there exists programs P^L and P^R over Σ-structures such that $P^L \in$BMRS^p, $P^R \in$BMRS^s, and for every $w \in \Sigma^*$, $P^L \circledcirc P^R$ yields a model of $f(w)$.

Note that a program which uses neither successor nor predecessor is in both BMRS^p and BMRS^s. The identity program in Proposition 5.5(c) is an example of such a program. An immediate consequence of this is that any function that can be modeled with a BMRS program that only uses successor (or only uses predecessor) is weakly deterministic. A right subsequential function, for example, can always be modeled by some program P^R in BMRS^s, and can therefore be modeled as $P^R \circledcirc P^L$, where P^L is the identity program over the same alphabet. Moreover, *bounded* circumambient patterns are weakly deterministic. Recall the epenthesis map $\emptyset \to b / a\_\_a$ from Example 2.12 in Section 2.2.3; this map requires information in two directions because an 'b' gets epenthesized if it is preceded by an 'a' *and* followed by an 'a'. However, the BMRS implementation in (13) only used the successor function. In general, any program which does not require recursion can be expressed as an equivalent program which uses only successor or uses only predecessor with the use of copy sets [Bhaskar et al., 2020]. This means that strictly local functions in general are also weakly deterministic. Further, if a program uses both predecessor and successor but only successor is used recursively, then it can be rewritten as a program which only uses successor (and similarly if only predecessor is used recursively). These remarks and the use of copy sets are not discussed in further detail in this section because they are not necessary for modeling the weakly deterministic processes presented here.

The remainder of this section shows how Definition 6.2 can be used to show that bidirectional harmony and default-to-same stress are weakly deterministic, and how it can be used to prove when functions are *not* weakly deterministic. The latter point is especially significant because Definition 6.2 characterizes weakly deterministic functions as those that can be modeled with programs embedded in BMRS^{p,s}. Sour Grapes, for example, can be modeled as a program within BMRS^{p,s}, but not as one one that can be expressed as the simultaneous application of programs in

BMRS^p and BMRS^s (to be presented in Theorem 6.3). Intuitively, this is because BMRS^p can only check for a trigger for harmony on the left while BMRS^s can only check for a blocker on the right. Since neither one on its own can determine whether a segment undergoes harmony, the simultaneous application of both cannot either. Simultaneous application of programs in BMRS^p and BMRS^s restricts the space of program in BMRS^{p,s} to those which do not require non-local information in *both* directions. Since BMRS^{p,s} programs characterize the rational functions, Definition 6.2 characterizes a *proper subclass* of the rationals. These observations show that Definition 6.2 is consistent with the Subregular Hierarchy in Figure 3.1. The relationship between the classes of functions in the subregular hierarchy and BMRS programs is summarized in Figure 6.4.

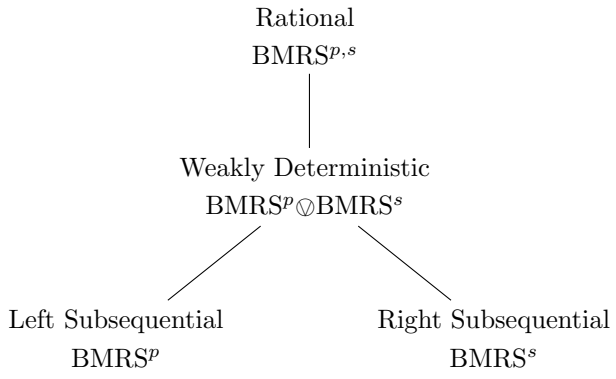


Figure 6.4: BMRS characterizations of long-distance maps (i.e. those that require recursion). BMRS^p⟡BMRS^s represents the BMRS^{p,s} programs which are expressed as the simultaneous application of programs in BMRS^p and BMRS^s.

6.2.1 Bidirectional Harmony

(66) presented data from a bidirectional pharyngeal harmony (emphasis spread) in Palestinian Arabic. Selected data points are repeated below in (71). This pattern can be decomposed into two contradirectional pharyngeal harmony patterns: leftward harmony in (a), and rightward harmony with a blocker in (b) and (c). When the underlying emphatic sound is in the middle of the word, bidirectional spreading takes place, as in (d). What makes this pattern weakly deterministic is its disjunctive nature; a segment in the input undergoes pharyngeal harmony if there is either a trigger for harmony on the right, or there is a trigger for harmony and no blocker on the left.

(71) *Pharyngeal harmony, selected data points from (66)*

a. /xayyaaṬ/ ↦ [XAYYAAṬ]

b. /Ṭuub-ak/ ↦ [ṬUUB-AK]

c. /Ṣayyad/ ↦ [SAyyad]

d. /ʔaṬfall/ ↦ [ʔAṬFALL]

Davis [1995] uses the feature [RTR] to represent emphatic or pharyngealized consonants. Jardine and Oakden [2023] present a BMRS implementation of this pattern, repeated here as (72). The input predicates RTR, HIGH, and FRONT evaluate to $\top$ when a segment in the input is emphatic, [+high], and [+front], respectively. Whether a segment get pharyngealized is determined by the output predicate RTR$'$. This function first checks if a segment is emphatic in the input; if $\text{RTR}(x) = \top$, then it gets pharyngealized (i.e. $\text{RTR}'(x) = \top$). Otherwise, the function checks for a trigger of harmony on the right. Since the pattern is iterative (similar to Example 2.10), the function checks for a pharyngeal trigger in the output rather than the input. If $\text{RTR}'(s(x)) = \top$, then leftward pharyngeal harmony takes place. If a trigger is not found on the right, then the function checks for a blocker.[1] If $\text{HIGH}(x)$ and $\text{FRONT}(x)$ both hold, then harmony is blocked and the function evaluates to $\bot$. If x is not a blocker for harmony, then the function checks for a trigger of harmony on the left, which corresponds with the last line of (72).

(72) *Pharyngeal Harmony, as a BMRS equation* [Jardine and Oakden, 2023]

$$\begin{aligned}\text{RTR}'(x) = \quad & \text{if } \text{RTR}(x) \text{ then } \top \text{ else} \\ & \text{if } \boxed{\text{RTR}'(s(x))} \text{ then } \top \text{ else} \\ & \text{if } \big(\text{HIGH}(x) \wedge \text{FRONT}(x)\big) \text{ then } \bot \text{ else} \\ & \boxed{\text{RTR}'(p(x))}\end{aligned}$$

Because this harmony pattern is bidirectional, RTR$'$ is a recursive function which iterates using both successor and predecessor, as highlighted in (72). However, for any input x only one of $\text{RTR}'(s(x))$ and $\text{RTR}'(p(x))$ determines the output value $\text{RTR}'(x)$. If $\text{RTR}'(s(x)) = \top$, then $\text{RTR}'(p(x))$ will never be evaluated. This happens in cases of leftward harmony. On the other hand, $\text{RTR}'(p(x))$ will only be evaluated if $\text{RTR}'(s(x)) = \bot$ (and $\text{HIGH}(x) \wedge \text{FRONT}(x) = \bot$), which happens in the case of rightward harmony (with no blocker). Thus, any segment that undergoes pharyngeal harmony is going to do so either because $\text{RTR}'(s(x)) = \top$ or because $\text{RTR}'(p(x)) = \top$. This is reminiscent of the fact that each segment undergoes harmony because of a trigger either to the left or to the right.

Pharyngeal harmony can be expressed as two contradirectional harmony maps: a right subsequential harmony map for (71a) and a left subsequential harmony map for (71b,c). The former is

[1] Recalling the discussion of licensing and blocking structures in Section 3.3, $\text{RTR}'(s(x))$ is a licensing structure while $(\text{HIGH}(x) \wedge \text{FRONT}(x))$ is a blocking structure.

computed by $\text{RTR}^R(x)$ in (73a), which only uses successor since it looks for a trigger of harmony on the right. The latter is computed by the function $\text{RTR}^L(x)$ in (73b). This function has an extra step since it must check whether the input x is a blocker of harmony. If $\textsc{Height}(x) \wedge \textsc{Front}(x) = \bot$, then this function checks for a trigger of harmony on the left, and therefore only uses predecessor.

(73) *Leftward and rightward pharyngeal harmony*

a. *Rightward pharyngeal harmony, as a BMRS^s equation*
$\text{RTR}^R(x) = \text{if RTR}(x) \text{ then } \top \text{ else } \text{RTR}^R(s(x))$

b. *Leftward pharyngeal harmony, as a BMRS^p equation*
$$\begin{aligned}\text{RTR}^L(x) = \quad & \text{if RTR}(x) \text{ then } \top \text{ else} \\ & \text{if } \big(\textsc{High}(x) \wedge \textsc{Front}(x)\big) \text{ then } \bot \text{ else} \\ & \text{RTR}^L(p(x))\end{aligned}$$

The simultaneous application of these two functions is presented in (74). The first line of $\text{RTR}^{L \circledcirc R}(x)$ is the definition of simultaneous application over two BMRS expressions, which was given in Definition 5.4. In the case where $\text{RTR}(x) = \top$, both $\text{RTR}^R(x)$ and $\text{RTR}^L(x)$ evaluate to $\top$ and therefore $\text{RTR}^R(x) \wedge \text{RTR}^L(x) = \top$. Thus, $\text{RTR}^{L \circledcirc R}(x)$ can be simplified to the equivalent expression in the second line of (74). Recall from (5) that $P(x) \vee Q(x)$ is shorthand for $\big[\text{if } P(x) \text{ then } \top \text{ else } Q(x)\big]$. Thus, the second and third lines of (74) are equivalent. The final line of (74) is the result of expanding the values of $\text{RTR}^R(x)$ and $\text{RTR}^L(x)$. These expression will only be evaluated when $\text{RTR}(x) = \bot$, in which case, $\text{RTR}^R(x)$ will always compute the 'else' part of (73a) and similarly, $\text{RTR}^L(x)$ will always compute the 'else' part of (73b). In other words, $\text{RTR}^R(x)$ will always compute $\text{RTR}^R(s(x))$ and $\text{RTR}^L(x)$ will always compute $\big[\text{if } \textsc{High}(x) \wedge \textsc{Front}(x) \text{ then } \bot \text{ else } \text{RTR}^L(p(x)\big]$. Thus the definition of $\text{RTR}^{L \circledcirc R}$ can equivalently be expressed as the final `if...then...else` expression in (74).

(74) *Simultaneous application of leftward and rightward harmony*
$$\begin{aligned}\text{RTR}^{L \circledcirc R}(x) \quad & = \text{if RTR}(x) \text{ then } \big(\text{RTR}^R(x) \wedge \text{RTR}^L(x)\big) \text{ else } \big(\text{RTR}^R(x) \vee \text{RTR}^L(x)\big) \\ & \equiv \text{if RTR}(x) \text{ then } \top \text{ else } \big(\text{RTR}^R(x) \vee \text{RTR}^L(x)\big) \\ & \equiv \text{if RTR}(x) \text{ then } \top \text{ else } \big(\text{if RTR}^R(x) \text{ then } \top \text{ else RTR}^L(x)\big) \\ & \equiv \text{if RTR}(x) \text{ then } \top \text{ else} \\ & \quad \text{if } \text{RTR}^R(s(x)) \text{ then } \top \text{ else} \\ & \quad \text{if } \big(\textsc{High}(x) \wedge \textsc{Front}(x)\big) \text{ then } \bot \text{ else} \\ & \quad \text{RTR}^L(p(x))\end{aligned}$$

The significance of rewriting the simultaneous application of the two BMRS expression in this equivalent form is that the last line of (74) looks exactly like the BMRS expression in (72). Simultaneous application of the two individual harmony functions indeed computes exactly the same function as the original bidirectional pattern. The distinction between the functions in (72) and (74) is that (74) makes explicit reference to two separate functions. Iterating to the left is always done using the RTR^L function while iterating to the right is always done using the RTR^R function. This fact is more concretely demonstrated in the computations in Figure 6.5, where (a) demonstrates the computation of leftward harmony, (b) demonstrates the computation of rightward harmony, (c) demonstrates rightward harmony with a blocker, and (d) represents bidirectional harmony. The details of how RTR' is evaluated over these input strings can be found in Jardine and Oakden [2023].

It is clear to see in these examples that the harmony pattern in (71a) is exclusively the result of changes made by RTR^R, as highlighted in the second row of the table in Figure 6.5(a). Similarly, the harmony pattern in (71b) is exclusively the result of changes made by RTR^L, as highlighted in third row of the table in Figures 6.5(b). The bidirectional example in (71d) is presented in Figure 6.5(c). This table illustrates how the simultaneous application operator over programs here captures precisely the same idea as μ-conserving compositions [Meinhardt et al., 2021] that were presented in Figure 6.3 in Section 6.1.2. The first two elements which undergo harmony are the result of RTR^R while the last four elements are the result of RTR^L. The overall bidirectional pattern is expressed as two separate contradirectional subsequential processes which run parallel, and each input-output correspondence is the result of one of these processes. These examples show that simultaneous application not only succeeds in capturing the bidirectional nature of this pattern, but also does so in a way that reflects why this pattern is weakly deterministic.

	x	a	y	y	a	ṯ
$\mathrm{RTR}(x)$	⊥	⊥	⊥	⊥	⊥	⊤
$\mathrm{RTR}^{R}(x)$	⊤	⊤	⊤	⊤	⊤	⊤
$\mathrm{RTR}^{L}(x)$	⊥	⊥	⊥	⊥	⊥	⊤
$\mathrm{RTR}^{R\circledcirc L}(x)$	⊤	⊤	⊤	⊤	⊤	⊤
	X	A	Y	Y	A	Ṭ

(a) Leftward harmony; input-output relation from (71a)

	ṯ	u	u	b	a	k
$\mathrm{RTR}(x)$	⊤	⊥	⊥	⊥	⊥	⊥
$\mathrm{RTR}^{R}(x)$	⊤	⊥	⊥	⊥	⊥	⊥
$\mathrm{RTR}^{L}(x)$	⊤	⊤	⊤	⊤	⊤	⊤
$\mathrm{RTR}^{R\circledcirc L}(x)$	⊤	⊤	⊤	⊤	⊤	⊤
	Ṭ	U	U	B	A	K

(b) Rightward harmony: input-output relation from (71b)

	ʔ	a	ṯ	f	a	l	l
$\mathrm{RTR}(x)$	⊥	⊥	⊤	⊥	⊥	⊥	⊥
$\mathrm{RTR}^{R}(x)$	⊤	⊤	⊤	⊥	⊥	⊥	⊥
$\mathrm{RTR}^{L}(x)$	⊥	⊥	⊤	⊤	⊤	⊤	⊤
$\mathrm{RTR}^{R\circledcirc L}(x)$	⊤	⊤	⊤	⊤	⊤	⊤	⊤
	ʔ	A	Ṭ	F	A	L	L

(c) Bidirectional harmony; input-output relation from (71d)

Figure 6.5: Pharyngeal harmony as the simultaneous application of the leftward harmony equation RTR^{L} and rightward harmony equation RTR^{R}.

Stem-Controlled Harmony

Stem-controlled vowel harmony [Baković, 2000, 2003, Krämer, 2003] is the phonological process by which the surface form of affixes in a word depend on the stem of the word. Within words which have both prefixes and suffixes, such as Akan ATR harmony in (18), the feature spreads *outward*, yielding a bidirectional harmony pattern similar to pharyngeal harmony in Palestinian Arabic. (75) presents a simplified version of stem-controlled harmony, where stems are assumed to always have a single vowel. For harmony process involving some feature F, + indicates a sound that is [+F] and − indicates a sound that is [-F], and $\sqrt{\cdot}$ indicates the stem of a word. A BMRS implementation of (75) is presented here to show why this simplified function is consistent with Definition 6.2 of weak determinism. More complex stem-controlled maps involving stems with multiple syllables and underspecified underlying feature specifications (as in the case of Akan) are discussed briefly but not pursued formally.

(75) *Simplified stem-controlled harmony map, where* $v, w \in \{+, -\}^*$

a. $v\sqrt{+}w \;\mapsto\; +^{|v|}\sqrt{+}+^{|w|}$

b. $v\sqrt{-}w \;\mapsto\; -^{|v|}\sqrt{-}-^{|w|}$

The BMRS implementation of (75) is defined in several steps. First, it is necessary to know at index x whether the stem of the word is to the left of x or to the right. This is equivalent to knowing whether x is a sound in the prefix or in the suffix. Determining this information requires two recursive functions, given in (76). The function Stem-L(x) in (76a), for example, evaluates to $\top$ if and only if the stem of the word is to the left of x. This equation is similar to the recursive `b-left` function in (12), which looks for a 'b' to the left of an index.

(76) *Determining whether the stem of the word is to the left/right of any index*

a. *Stem is to the left of* x

Stem-L(x) = if Initial(x) then $\bot$ else
if Stem(px) then $\top$ else
Stem-L(px)

b. *Stem is to the right of* x

Stem-R(x) = if Final(x) then $\bot$ else
if Stem(sx) then $\top$ else
Stem-R(sx)

We also need to define a function which yields the feature specification of the stem. That is, we need a function that evaluates to $\top$ if and only if the stem vowel is [+F]. Since the ultimate goal here is to split stem-controlled harmony into two separate programs, this function is expressed in two parts in (77). If the stem is found to the left of x, the function F-Stem-L(x) in (77a) searches through indices $y < x$ for the stem, and returns $\top$ iff F holds at the index which carries the stem. If the stem is *not* found to the left of x, the function will eventually reach the first index of the string and return $\bot$. Thus, the function will evaluate to $\bot$ for all indices to the left of the stem. This is to ensure that the function terminates (i.e. assigns a truth value at every index). In practice, however, this function will only be called for indices where Stem-L holds. F-Stem-R(x) is defined similarly for words where the stem is to the right of x.

(77) *Feature value of stem;* F-Stem(x) = $\top$ *iff the stem is* [+F]

a. *Return feature value of stem, if it is to the left of* x*;* $\bot$ *otherwise*

F-Stem-L(x) = if Stem(x) then $F(x)$ else
if Initial(x) then $\bot$ else
F-Stem-L(px)

b. *Return feature value of stem, if it is to the right of x; $\bot$ otherwise*

$$\begin{aligned}\text{F-Stem-R}(x) = \;& \text{if Stem}(x) \text{ then } F(x) \text{ else}\\ & \text{if Final}(x) \text{ then } \bot \text{ else}\\ & \text{F-Stem-R}(sx)\end{aligned}$$

The leftward and rightward harmony maps are presented in (78) for an arbitrary feature F that undergoes harmony. Consider the function $\text{F}^L(x)$; if the stem of the word is found to the left of x, this function to $\top$ iff the stem is [+F]; otherwise, if the stem is not to the left, the function makes no changes. Thus, $\text{F}^L(x)$ yields rightward stem-controlled harmony, where the value for [F] spreads from the stem to the suffixes only. In a similar manner, F^L yields leftward stem-controlled harmony, spreading the stem value to the prefixes. The simultaneous application of these two processes over the strings $--\sqrt{+}--$ and $++\sqrt{-}++$ are presented in Figure 6.6.

(78) *Leftward and rightward stem-controlled harmony*

a. *Harmony from stem to suffixes, as a BMRS^L equation*

$$\begin{aligned}\text{F}^L(x) = \;& \text{if Stem}(x) \text{ then } F(x) \text{ else}\\ & \text{if Stem-L}(x) \text{ then F-Stem}^L(x) \text{ else}\\ & \text{F}(x)\end{aligned}$$

b. *Harmony from stem to prefixes, as a BMRS^R equation*

$$\begin{aligned}\text{F}^R(x) = \;& \text{if Stem}(x) \text{ then } F(x) \text{ else}\\ & \text{if Stem-R}(x) \text{ then F-Stem}^R(x) \text{ else}\\ & \text{F}(x)\end{aligned}$$

	−	−	$\sqrt{+}$	−	−
F(x)	⊥	⊥	⊤	⊥	⊥
Stem(x)	⊥	⊥	⊤	⊥	⊥
Stem-R(x)	⊤	⊤	⊥	⊥	⊥
Stem-L(x)	⊥	⊥	⊥	⊤	⊤
F-Stem^R(x)	⊤	⊤	⊤		
F-Stem^L(x)			⊤	⊤	⊤
F^R(x)	⊤	⊤	⊤	⊥	⊥
F^L(x)	⊥	⊥	⊤	⊤	⊤
$F^{R\mathbin{⩒}L}$(x)	⊤	⊤	⊤	⊤	⊤
	+	+	$\sqrt{+}$	+	+

(a) [+F] root

	+	+	$\sqrt{-}$	+	+
F(x)	⊤	⊤	⊥	⊤	⊤
Stem(x)	⊥	⊥	⊤	⊥	⊥
Stem-R(x)	⊤	⊤	⊥	⊥	⊥
Stem-L(x)	⊥	⊥	⊥	⊤	⊤
F-Stem^R(x)	⊥	⊥	⊥		
F-Stem^L(x)			⊥	⊥	⊥
F^R(x)	⊥	⊥	⊥	⊤	⊤
F^L(x)	⊤	⊤	⊥	⊥	⊥
$F^{R\mathbin{⩒}L}$(x)	⊥	⊥	⊥	⊥	⊥
	-	-	$\sqrt{-}$	-	-

(b) [-F] root

Figure 6.6: Stem-controlled harmony as the simultaneous application of leftward and rightward stem-controlled harmony. F^R spreads the feature F from the stem to the left and F^L spreads the feature from the stem to the right (highlighted in rows 7 and 8). The simultaneous application $F^{R\mathbin{⩒}L}$ spreads the feature outward, yielding bidirectional harmony similar to pharyngeal harmony.

Section 3.1.2 presented stem-controlled harmony in Akan as an example of a weakly deterministic function. The complication with Akan stem-controlled harmony is that the vowels in the affixes are unspecified for [ATR]. Two data points with their underlying forms are are repeated below in (79). The third person singular (3S) prefix /O-/ is neither [+ATR] nor [-ATR] underlyingly. In (79a), it surfaces as the [-ATR] vowel [ɔ-] because the stem vowel is [-ATR]; in (79b), it surfaces as the the [+ATR] vowel [o-] because the stem vowel is [+ATR].

(79) Stem-Controlled ATR Harmony, selected data points from (18)

a. /O-bE-tʊ-I/ [**ɔ**-b**ɛ**-t**ʊ̲**-**ɪ**] '3S-FUT-throw-3S.OBJ'
b. /O-bE-tu-I/ [**o**-b**e**-t**u̲**-**i**] '3S-FUT-dig-3S.OBJ

The complication with this example is that the definition of simultaneous application assumes that input strings are specified for each of the features. Thus, while the demonstration above shows why stem-controlled harmony is in general weakly deterministic, the programs in (78) do not apply neatly to the case of Akan. In order to account for a larger range of linguistic processes, we need a way to model under-specification in BMRS. This topic is beyond the scope of this chapter. Further discussion of underspecification in BMRS can be found in Nelson and Baković [2024]

6.2.2 Default-to-Same Stress

LHOL stress was presented in (67) with sample data from Murik. The general form of LHOL is presented in (80) as a map over $\{H, L, \acute{H}, \acute{L}\}$. Similar to pharyngeal harmony, this function can be decomposed into two independent functions where one looks to the left and the other looks to the right. In particular, LHOL can be decomposed into a function which stresses the leftmost heavy syllable and another function which stresses the leftmost light syllable if a heavy one does not exist. These are presented in (67a) and (67b), respectively. The function which stresses the leftmost heavy syllable must be able to check that there are no other heavy syllables to the left, and the function which stresses the leftmost light syllable must be able to check that there are no heavy syllables to the right.

(80) *LHOL stress, as a map over* $\{H, L, \acute{H}, \acute{L}\}^*$

a. $\mathrm{L}^n\mathrm{H}w \mapsto \mathrm{L}^n\acute{\mathrm{H}}w$ for $n \geq 0$ and $w \in \{H, L\}^*$ (*Stress Leftmost heavy syllable*)
b. $\mathrm{L}^n \mapsto \acute{\mathrm{L}}\mathrm{L}^{n-1}$ for $n > 0$ (*Default to leftmost if no heavy syllables*)

A BMRS program that models LHOL stress will have predicates $\{H, L, \textsc{Stress}, \textsc{Initial}, \textsc{Final}\}$. $H(x) = \top$ iff x is a heavy syllable and similarly, $L(x) = \top$ iff x is a light syllable. We assume that syllables are not stressed underlyingly, and set $\textsc{Stress}(x) = \bot$ for every x. We also define two auxiliary recursive functions (81a) and (81b); for a syllable x, NoH-L(x) evaluates to $\top$ when there are no heavy syllables to the *left* of x and NoH-R(x) evaluates to $\top$ when there are no heavy syllables to the *right* of x. These functions are very similar to the general function `b-left` which was defined in (11) of Section 2.2.2 to show how recursive equations simulate quantifiers. The equation for NoH-L(x), for example, expresses $\neg\exists y(y < x \wedge H(y))$. Koser and Jardine [2020] give an implementation of LHOL stress in BMRS, repeated below in (82).[2]

(81) *Recursive BMRS functions which check for H syllables*

a. *No heavy syllables to the left of x*

$$\begin{aligned}\text{NoH-L}(x) = \quad & \text{if } \textsc{Initial}(x) \text{ then } \top \text{ else} \\ & \text{if } H(p(x)) \text{ then } \textit{bot} \text{ else} \\ & \text{NoH-L}(p(x))\end{aligned}$$

b. *No heavy syllables to the right of x*

$$\begin{aligned}\text{NoH-R}(x) = \quad & \text{if } \textsc{Final}(x) \text{ then } \top \text{ else} \\ & \text{if } H(s(x)) \text{ then } \textit{bot} \text{ else} \\ & \text{NoH-R}(s(x))\end{aligned}$$

(82) *LHOL stress as a BMRS equation* [Koser and Jardine, 2020]

$$\textsc{Stress}'(x) = \big(L(x) \wedge \textsc{Initial}(x) \wedge \text{NoH-R}(x)\big) \vee \big(H(x) \wedge \text{NoH-L}(x)\big)$$

Recall from Section 3.1.2 that LHOL stress can be decomposed into two subsequential processes: a left subsequential process which stresses the leftmost heavy syllable, and a right subsequential function which stresses the leftmost light syllable in a word with no heavy syllables. These two processes are modeled by the two output predicates $\textsc{Stress}^L$ and $\textsc{Stress}^R$ in (83). $\textsc{Stress}^L(x) = \top$ exactly when x is a heavy syllable and there are no heavy syllables to the left of x. In a word with no heavy syllables, $\textsc{Stress}^L(x)$ does not stress any syllables. $\textsc{Stress}^R(x) = \top$ exactly when x is an initial light syllable, and there are no heavy syllables anywhere to the right of x.

[2]The original BMRS definition given by Koser and Jardine [2020] uses the functions PRECEDE-H and FOLLOW-H, and is defined as follows:

$$\textsc{Stress}'(x) = \big(L(x) \wedge \textsc{Initial}(x) \wedge \neg\textsc{Precede-H}(x)\big) \vee \big(H(x) \wedge \neg\textsc{Follow-H}(x)\big)$$

The definition of $\textsc{Stress}'(x)$ in (82) is equivalent to the definition given by Koser & Jardine. The function NoH-L for example, is equivalent to ¬PRECEDE-H.

(83) *Two independent stress-assigning functions*

a. *Stress the leftmost heavy syllable*
$\textsc{Stress}^L(x) = H(x) \wedge \textsc{noH-L}(x)$

b. *Stress the leftmost light syllable if a word has no heavy syllables*
$\textsc{Stress}^R(x) = L(x) \wedge \textsc{Initial}(x) \wedge \textsc{NoH-R}(x)$

$\textsc{Stress}^L$ is defined in terms of the recursive function noH-L, and therefore only uses the predecessor function. In other words, $\textsc{Stress}^L \in \text{BMRS}^p$. Similarly, $\textsc{Stress}^R \in \text{BMRS}^s$. The simultaneous application of these two functions is given in (84). Since we assume that $\textsc{Stress}(x) = \bot$ for all x, the 'if' part of the expression will always evaluate to $\bot$. This then means that the 'then' part of the $\textsc{Stress}^{L \circledcirc R}$ will never be evaluated. Thus, the entire expression simplifies to the 'else' part in the final line of (84). This expression is exactly the the stress-assignment function that was presented in (82); this is clear to see by substituting the expressions for $\textsc{Stress}^R(x)$ and $\textsc{Stress}^L(x)$ into the disjunction in (84). Thus, $\textsc{Stress}^{L \circledcirc R} \equiv \textsc{Stress}'$.

(84) *Simultaneous application of the two stress-assignment functions*

$$\begin{aligned}
\textsc{Stress}^{R \circledcirc L}(x) &= \text{if } \textsc{Stress}(x) \text{ then } \big(\textsc{Stress}^R(x) \wedge \textsc{Stress}^L(x)\big) \text{ else} \\
&\quad \big(\textsc{Stress}^R(x) \vee \textsc{Stress}^L(x)\big) \\
&\equiv \text{if } \bot \text{ then } \big(\textsc{Stress}^R(x) \wedge \textsc{Stress}^L(x)\big) \text{ else } \big(\textsc{Stress}^R(x) \vee \textsc{Stress}^L(x)\big) \\
&\equiv \textsc{Stress}^R(x) \vee \textsc{Stress}^L(x)
\end{aligned}$$

The computation of $\textsc{Stress}^{f \circledcirc g}$ over the strings 'LLLL' and 'LHLLH' is presented in Figure 6.7. (a) shows that when a string has only light syllables, the placement of stress is determined exclusively by $\textsc{Stress}^L$. Similarly, (b) shows that when a string has heavy syllables, placement of stress is determined exclusively by $\textsc{Stress}^R$.

The BMRS program for LHOL stress can easily be extended to RHOR (rightmost-heavy-otherwise-right) stress. RHOR is presented in (85); the rightmost heavy syllable in a word get stressed, and the rightmost light syllable gets stressed in a word that does not have any heavy syllables. The BMRS equation for RHOR stress is presented in (86), along with a decomposition into $\textsc{Stress}^L$ and $\textsc{Stress}^R$. This equation is notably very similar to the BMRS equation for LHOL stress in (82). It is therefore easy to see that RHOR stress is also weakly deterministic.

	L	L	L	L
NoH-L(x)	⊤	⊤	⊤	⊤
NoH-R(x)	⊤	⊤	⊤	⊤
Stress^L(x)	⊤	⊥	⊥	⊥
Stress^R(x)	⊥	⊥	⊥	⊥
Stress$^{L⊛R}$(x)	⊤	⊥	⊥	⊥
	Ĺ	L	L	L

(a) Stressing leftmost L

	L	H	L	L	H
NoH-L(x)	⊤	⊤	⊥	⊥	⊥
NoH-R(x)	⊥	⊥	⊥	⊥	⊤
Stress^L(x)	⊥	⊥	⊥	⊥	⊥
Stress^R(x)	⊥	⊤	⊥	⊥	⊥
Stress$^{L⊛R}$(x)	⊥	⊤	⊥	⊥	⊥
	L	Ḧ	L	L	H

(b) Stressing leftmost H

Figure 6.7: LHOL stress as the simultaneous application application of Stress^L and Stress^R. If a word only has light syllables, Stress^L determines which syllable to stress, highlighted in (a). If a word has heavy syllables, Stress^R determines which syllable to stress, highlighted in (b).

(85) *RHOR stress, as a map over* $\{H, L, \acute{H}, \acute{L}\}$

a. $w\text{HL}^n \mapsto w\acute{\text{H}}\text{L}^n$ for $n \geq 0$ and $w \in \{H, L\}^*$ (*Stress rightmost heavy syllable*)

b. $\text{L}^n \mapsto \text{L}^{n-1}\acute{\text{L}}$ for $n > 0$ (*Default to rightmost if no heavy syllables*)

(86) *RHOR stress as a BMRS equation, and decomposition into* Stress$^L(x)$ *and* Stress$^R(x)$

$$\textsc{Stress}'(x) = \underbrace{\big(L(x) \wedge \textsc{Final}(x) \wedge \textsc{NoH-L}(x)\big)}_{\textsc{Stress}^L(x)} \vee \underbrace{\big(H(x) \wedge \textsc{NoH-R}(x)\big)}_{\textsc{Stress}^R(x)}$$

Figures 6.5 and 6.7 highlight how LHOL stress and pharyngeal harmony are similar. In both cases, the overall map is bidirectional and a BMRS implementation requires recursion with both predecessor and successor. However, both can be decomposed into a map which only uses successor and a map which only uses predecessor in such a way that *every instance of change made to an input string is associated exclusively with one of the functions in the decomposition.* This is exactly the property that makes them weakly deterministic. Definition 6.2 of weakly deterministic function in terms of simultaneous application therefore succeeds in encapsulating the informal concept of a weakly deterministic function.

6.2.3 Non-Weakly Deterministic Maps

This section shows that Sour Grapes harmony in (68), Yidiny liquid dissimilation in (19), Copperbelt Bemba high tone spreading in (69), and Tutrugbu ATR harmony in (70) are not weakly deterministic with respect to Definition 6.2. Sour Grapes harmony is repeated here as the general map over $\{-, +\}^*$ in (87). This pattern involves some feature (marked $+$) that can only spread through a word if there isn't a blocker (marked $\boxminus$) later in the word. Theorem 6.3 shows that

the definition of weakly deterministic functions in terms of simultaneous application succeeds in excluding Sour Grapes from the class of weakly deterministic functions.

(87) *Sour Grapes as a map over* $\{+,-\}^*$

a. $+-^n \mapsto\ ++^n$

b. $+\,-^n \boxminus \mapsto\ +\,-^n \boxminus$

c. $-^n \mapsto\ -^n$

Theorem 6.3. Sour grapes is not weakly deterministic.

Proof. Consider the Sour Grapes pattern as a string function over the alphabet $\Sigma = \{a, b, c\}$, where a 'b' represents a segment with the feature that is spreading rightwards, 'a' is the segment without the feature (i.e. target of harmony), and 'c' is the blocker. Assume this function is weakly deterministic. Then by Def. 6.2 there are two BMRS programs P^R and P^L such that the functions in P^R use only successor, the functions in P^L use only predecessor, and $P^L \mathbin{⦶} P^R$ models Sour Grapes harmony. This means $P^L \mathbin{⦶} P^R$ yields the string transformations in (88a-c) for all $m, n \geq 0$.

(88) *Sour Grapes as a map over* $\{a, b, c\}^*$

a. $\boldsymbol{b}a^m\ a\ a^n a \ \mapsto\ \boldsymbol{b}b^m\ b\ b^n b$

b. $\boldsymbol{b}a^m\ a\ a^n\boldsymbol{c} \ \mapsto\ \boldsymbol{b}a^m\ a\ a^n\boldsymbol{c}$

c. $aa^m\ a\ a^n a \ \mapsto\ aa^m\ a\ a^n a$

We consider first the case where there are no copy sets, and consider any $m, n \geq 0$. P^R must have the output predicate b^R, P^L must have the output predicate b^L and the simultaneous application $b^{L⦶R}$ must assign the following truth values for the input string in (a).

input	b	a^m	a	a^n	a
index	0	$\cdots$	m	$\cdots$	$m+n+1$
$b(x)$	$\top$	$(\bot)^m$	$\bot$	$(\bot)^n$	$\bot$
$b^{R⦶L}(x)$	$\top$	$(\top)^m$	$\top$	$(\top)^n$	$\top$
output	b	b^m	b	b^n	b

Since $b(m) = \bot$ and $b^{R⦶L}(m) = \top$ over the input string '$\mathrm{ba}^m\mathrm{aa}^n\mathrm{a}$', it follows from (5.10) in Proposition 5.16 that either $b^R(m) = \top$ or $b^L(m) = \top$ (over the string '$\mathrm{ba}^m\mathrm{aa}^n\mathrm{a}$'). In other words, because $b^{L⦶R}$ flips the truth value associated with b at index m, it must be the case that either b^L or b^R flips the truth value when evaluated over the same input.

Consider the first case. Because b^L uses only predecessor, it cannot distinguish between the input strings in (a) and (b) at index m (highlighted). If $b^L(m) = \top$ over the input string 'ba^m a a^{n}a' then it must also evaluate to $\top$ for the string 'ba^m a a^{n}c', and therefore $b^{R \otimes L}(m) = \top$ over the input string 'ba^m a a^{n}c'. This means that the output string for 'ba^m a a^{n}c' will carry a 'b' in index m, contradicting the input-output relation in (b). Similarly in the second case, because b^R uses only successor, it cannot distinguish between the input strings in (a) and (c) at index m. Thus, if $b^R(m) = \top$ over the string '$b(a)^n$ a $(a)^m a$' then it must evaluate to $\top$ for the string '$a(a)^m$ a $(a)^n a$', and therefore $b^{R \otimes L}(m) = \top$ over the input string 'aa^m a $a^n a$'. This means the output string for 'aa^m a $a^n a$' will carry a 'b' at index m, contradicting the input-output relation in (c).

Thus, there is no way to capture the three maps in (a-c) which characterize Sour Grapes as simultaneous application in the case of no copy sets. Consider the case where $P^R \otimes P^L$ has k copy sets. Then we can take any $m, n > k$. Consider the highlighted element at index m in the output string '$\boldsymbol{b}(b)^m$ b $(b)^n b$'. Given the ordering on indices with a copy parameter (Figure 2.19), at most the first k output elements can be at index 0 and at most the last k output elements can be at index $m + n + 1$. Thus the highlighted output at index m cannot be at any index $(i, 0)$ or $(i, m + n + 1)$ for any $i \leq k$. Thus, there must be some index (i, x) where $0 < x < m + n + 1$ such that $b^{R \otimes L}(i, x) = \top$. Because $b(x) = \bot$ for any such x in the input string in (a), it must be the case that either $b^R(i, x) = \top$ or $b^R(i, x) = \top$. For any $0 < x < m + n + 1$, and $b^L(i, x)$ cannot distinguish between the input strings in (a) and (b), $b^R(i, x)$ cannot distinguish between the input strings in (a) and (c). Thus, the above argument holds in this case and leads to contradiction. □

The proof that Sour Grapes is not weakly deterministic gives a systematic way to show that unbounded circumambient functions are not weakly deterministic. The proof method involves breaking the map into three characteristic input-output relations in (88a-c) with the following conditions: (a) involves some a change at index m that requires information from the left and the right; (b) is identical to (a) from the beginning of the string up to index m but the input at index m doesn't undergo any change; (c) is identical to (a) from the index m to the end of the string but the input at index m doesn't undergo any change. If the map in (a) can be modeled by simultaneous application, then the change observed at index m should be the result either a program that only uses successor or a program that only uses predecessor. From there, the maps in (b) and (c) allow

us to derive a contradiction. Using this method, we can show that liquid dissimilation in Yidiny, high tone spreading in Copperbelt Bemba, and ATR harmony in Tutrugbu are also not weakly deterministic.

Theorem 6.4. Liquid dissimilation in Yidiny is not weakly deterministic.

Proof. Liquid dissimilation in Yidiny is characterized by the map in (89), where /l/ and /r/ are separated by an arbitrary number of segments that do not play any role in the dissimilation pattern.

(89) a. $x^m\, l\, x^n l \;\mapsto\; x^m\, r\, x^n l$
b. $x^m\, l\, x^n \;\mapsto\; x^m\, \mathrm{l}\, x^n$
c. $r x^{m-1}\, l\, x^n l \;\mapsto\; r x^{m-1}\, l\, x^n l$

The highlighted element at index m in the input string '$x^m\, l\, x^n l$' undergoes dissimilation. The input strings '$x^m\, l\, x^n l$' in (a) and $x^m\, l\, x^n$ in (b) are identical from the beginning of the string up to index m. The input string in (a) and $r x^{m-1}\, l\, x^n l$ in (c) are identical from index m up to the end of the string. Dissimilation takes place at index m in (a), but not in (b) or (c). The same argument as Theorem 6.3 therefore holds here; the simultaneous application of two programs where one only uses successor and the other only uses predecessor cannot model the transformations in (a-c) for all $m, n \geq 0$ because any such program would also have to yield dissimilation for either the input string in (b) or the input string in (c). □

Theorem 6.5. High tone spreading in Copperbelt Bemba is not weakly deterministic.

Proof. High tone spreading in Copperbelt Bemba is characterized by the maps in (90).

(90) a. $HL^m\, L\, L^n \;\mapsto\; HH^m\, H\, H^n$
b. $HL^m\, L\, L^n H \;\mapsto\; HHHL^{m-2}\, L\, L^n H$
c. $L^m\, L\, L^n \;\mapsto\; L^m\, L\, L^n$

The highlighted element at index m in the input string '$HL^m\, L\, L^n$' in (a) undergoes a change. The input strings '$HL^m\, L\, L^n$' in (a) and $HL^m\, L\, L^n H$ in (b) are identical from the beginning of the string up to index m. The input string in (a) and $L^m\, L\, L^n$ in (c) are identical from index m up to the end of the string. The element at index index m becomes a high tone in (a), but not in (b) or (c). The same argument as Theorem 6.3 therefore holds here; the simultaneous application of two programs where one only uses successor and the other only uses predecessor cannot model

the transformations in (a-c) for all $m, n \geq 0$ because any such program would also have to yield a high tone at index m for either the input string in (b) or the input string in (c). □

The example of ATR harmony in Tutrugbu (70) is similar to all the previously-discussed examples with the exception that it has a conditional blocker. [−high] vowels block ATR harmony with the root, but only when the initial syllable has a [+high] vowel. A schematic form of this map is presented in (91).

(91) *Tutrugbu ATR harmony as a map, where H represents a '−' that is also [+high] and H^* represents a '+' that is also [+high].*

a. $-^n+ \quad \mapsto \quad +^{n+1}$

b. $H\,-^n+ \quad \mapsto \quad H^+{+}^{n+1}$

c. $H\,-^m \boxminus -^n+ \quad \mapsto \quad H\,-^m \boxminus +^{n+1}$

d. $-^m \boxminus -^n+ \quad \mapsto \quad +^{m+n+1}$

Theorem 6.6. Tutrugbu ATR harmony is not weakly deterministic.

Proof. Consider the three maps in (92), where + and − refer to the [ATR] value of a vowel, the rightmost + or − represents the ATR value of the root vowel, and ⊟ represents a conditional blocker which only blocks right-to-left harmony when the initial vowel is [+high]. These maps represent a fragment of ATR harmony in Tutrugbu.

(92) a. $[-\text{high}](-)^m - (-)^n \boxminus \cdots + \quad \mapsto \quad [-\text{high}](+)^m + (+)^n \boxminus \cdots +$

b. $[-\text{high}](-)^m - (-)^n \cdots - \quad \mapsto \quad [-\text{high}](-)^m - (-)^n \cdots -$

c. $[+\text{high}](-)^m - (-)^m \boxminus \cdots + \quad \mapsto \quad [+\text{high}](-)^m - (-)^m \boxminus \cdots +$

The highlighted element at index m in the input string '$[-\text{high}](-)^m - (-)^n \boxminus \cdots +$' in (a) becomes [+ATR] because of the trigger for harmony on the right edge of the string. The input strings in (a) and (b) are identical from the beginning of the string up to index m. The input string in (a) and (c) are identical from index m up to the end of the string. The element at index index m becomes [+ATR] in (a), but not in (b) or (c). The same argument as Theorem 6.3 therefore holds here; the simultaneous application of two programs where one only uses successor and the other only uses predecessor cannot model the transformations in (a-c) for all $m, n \geq 0$ because any such program would also have to yield a [+ATR] value at index m for either the input string in (b) or the input string in (c). □

6.3 Comparison to Previous Proposals

Section 6.1.2 briefly introduced previous proposals for a formal characterization of the weakly deterministic functions. This section presents these proposals in more detail, with a discussion of why each of these definitions has not successfully captured the correct characterizations that were presented in Figure 6.1. While Definition 6.2 captures exactly the same intuitions and intention as these proposed definitions, their implementation within the framework of BMRS makes it possible reason about what kinds of patterns are and are not weakly deterministic.

6.3.1 No Mark-Up (Heinz & Lai, 2013)

The first formal definition of weakly deterministic functions was given by [Heinz and Lai, 2013]. Their definition, presented here as Definition 6.7, places two constraints on compositions of contradirectional subsequential functions; they must be alphabet-preserving and length-preserving. The first constraint is reflected in the requirement that domain and co-domain of the inner function of the composition be the same. This ensures that the inner function cannot introduce any new symbols that may be used to encode information about the input string. Since combinations of symbols in the original alphabet can also be used to encode information, Heinz & Lai add the length-preserving constraint in order to rule out this kind of encoding.

Definition 6.7 (**Proposal 1: No Mark-Up**)**.** [Heinz and Lai, 2013] A regular function τ is weakly deterministic iff there exists a left subsequential function $L : X^* \to X^*$ and a right subsequential function $R : X^* \to X^*$ such that L is not length-increasing and $\tau = R \circ L$.

Figure 6.2 in Section 6.1.2 showed how Sour Grapes harmony can be modeled as the composition of a left and right subsequential function using a special symbol that is used to encode information about whether the left side of the input string contains a trigger for harmony. Figure **??** shows how high tone spreading in Copperbelt Bemba from (69) can also be modeled as a composition of subsequential functions in the same way. The general form of the high tone spreading map is presented in (93); in (a) a high tone spreads to the end of the word if there are no other high tones to the right in the underlying form; in (b) a high tone only spreads to the next two syllables when another high tone is present to block unbounded spreading.

(93) *High tone spreading in Copperbelt Bemba, as a map over* $\{H, L\}^*$

a. $HL^n \mapsto HH^n$

b. $HL^nH \mapsto HHHL^{n-2}H$

High tone spreading can be modeled as the composition of a left and right subsequential function, as shown in Figure 6.8. The left subsequential function (LS) spreads a high tone (H) to the next two elements, and marks all remaining L tones with Ψ. This special symbol encodes information about the left side of the string, namely that there is an H tone somewhere to the left. The right subsequential function (RS) then determines whether Ψ should be an H or L tone based on information to the right. In (a), the right subsequential function turns Ψ into H tones because there is no H found on the right side of the string, while in (b) it turns Ψ into L because the H tone at the end of the string blocks further spreading. Even though RS only uses information found to the right of any input element, the intermediate representation allows it to determine the outputs for elements in the input which require non-local information in both directions. The intention behind Definition 6.7 is to place a constraint against this kind of encoding (called mark-up).

input	H	L	L	L	L
LS	H	H	H	Ψ	Ψ
RS	H	H	H	H	H

(a)

input	H	L	L	L	H
LS	H	H	H	Ψ	H
RS	H	H	H	L	H

(b)

Figure 6.8: High tone spreading in Copperbelt Bemba as the composition of left and right subsequential functions with intermediate mark-up [McCollum et al., 2020]

Heinz & Lai show that stem-controlled harmony and dominant-recessive harmony patterns are weakly deterministic with respect to Definition 6.7. However, this definition fails to exclude unbounded circumambient patterns from the weakly deterministic class. A simple case of this is liquid dissimilation in Yidiny, which is an unbounded circumambient pattern, but satisfies Definition 6.7 [Payne, 2017]. The two characteristic properties of this pattern are repeated in (94); in (a) an /l/ surfaces as [r] when there is another /l/ to the right of it, and in (b) this process gets blocks by an /r/ to the left. As discussed in Section 6.1 this pattern is unbounded circumambient because the /l/ in the 'going' affix which undergoes change needs non-local information in both directions: /l/ surfaces as [r] if and only if there is an /l/ to the right and no /r/ to the left.

(94) a. /l...l/ ↦ [r...l]
b. /r...l...l/ ↦ [r...l...l]

However, this pattern be modeled as 'double dissimilation' [Dixon, 1977]; a right subsequential dissimilation process (RS) turns an /l/ into an [r] if there is an /l/ to the right, and a left subsequential dissimilation process (LS) turns an /r/ into an [l] if there is another /r/ to the left. The composition of these two processes yields the blocking pattern in (94b), as shown in Figure 6.9. Notably, RS is length-preserving and does not introduce any new symbols. Thus, this pattern is weakly deterministic with respect to Definition 6.7, and points to an interesting loophole in Heinz & Lai's definition. The RS process posited by Dixon [1977] as a way to explain the pattern in (94) takes place *only* in the specific situation where the 'going' affix has undergone dissimilation, and reverts it back to an [l]. This process does not appear to take place in any other environment in Yidiny.[3] The intermediate 'r' in the 'going' affix therefore encodes information about there being an 'l' to the right; the affix has an 'r' because there must have been an /l/ that triggered dissimilation. LS can then use this information to turn 'l' back into 'r' in the presence of a blocker on the left. In this way, the intermediate 'r' in Figure 6.9 plays the same role as the special mark-up symbol Ψ in Figure **??**. In other words, double dissimilation is a form of mark-up that bypasses the constraints in Definition 6.7.

input	r	...	l	...	l
RS	r	...	r	...	l
LS	r	...	l	...	l

Figure 6.9: Liquid dissimilation in Yidiny as double dissimilation [Dixon, 1977]. The right subsequential dissimilation process (RS: $l \to r / \_\_ x^* l$) gives the intermediate form |r...r...l|. The left subsequential dissimilation process (LS: $r \to l / l x^* c \_\_$ then gives the surface form [r...l...l].

Subsequent work has shown that other unbounded circumambient patterns can also be modeled as the composition of left and right subsequential functions which bypass the constraints in Definition 6.7. Meinhardt et al. [2021] present two bidirectional dominant-recessive ATR harmony patterns in Turkana [Dimmendaal, 1983] and Massai [Tucker and Mpaayei, 1955] which exemplify the distinction between weakly deterministic and unbounded circumambient patterns. They show

[3]The details of this point were previously presented by Yolyan & McCollum at the 2021 Manchester Phonology Meeting.

that despite Massai ATR harmony being weakly deterministic and Turkana ATR harmony being unbounded circumambient, both get characterized as weakly deterministic with respect to Definition 6.7. This result is the motivation for their revised definition of weak determinism presented below as Definition 6.10. Further work showed that the unbounded circumambient patterns discussed in Section 6.1 are also weakly deterministic with respect to Definition 6.7. McCollum et al. [2018] show that Definition 6.7 characterizes ATR harmony in Tutrugbu and high tone spreading in Copperbelt Bemba as weakly deterministic. In particular, they show that the surface phonotactics of these languages can be exploited as a form of mark-up that is not ruled out by the conditions in Definition 6.7. For the above example of high tone spreading, they define a left subsequential function which writes an 'HL' at the end of a string that has an H tone to the left, and they do so without increasing the length of the string. The HL sequence encodes the same information as Ψ in Figure 6.8: that there is a trigger for tone spreading in the left side of the string. McCollum et al. [2018, pg.18] explain: "This word-final HL sequence can accomplish this because there is no other input that would generate a final sequence of HL. Since H spreads rightward and is only blocked by a following H, this substring is ill-formed". By using ill-formed substrings as mark-up, Copperbelt Bemba high tone spreading and Tutrugbu ATR harmony both bypass the constraints in Definition 6.7 and therefore get wrongly characterized as weakly deterministic. Lamont [2019] uses this method to show that Definition 6.7 also fails to exclude Sour Grapes harmony.

6.3.2 Mutation Maps (Meinhardt et. al 2021)

Meinhardt et al. [2021] proposed a new definition that relies on the concept of mutation maps, which represent the changes that a function induces in a string-to-string function. The definition of a mutation point over a *fixed input string* was presented in Chapter 5 as Definition 5.10. This definition is extended to the definition of a mutation *map* over a fixed string in Definition 6.8. Figure 6.3 showed how the mutation map of bidirectional emphasis spread in Palestinian Arabic over the input string /ʔaṬfall/ can be split up into the mutation map of a left subsequential function and a right subsequential function over the same string. This is the property which makes the pattern weakly deterministic. In other words, Meinhardt et al. [2021] argue that a string function f is weakly deterministic if it can be expressed as the composition of a left and right subsequential function such that every change made by the composition is a change made either by the left

subsequential function or the right subsequential function. This property is called 'μ-conserving', which was defined over mutation *points* in Section 5.2. The equivalent concept for mutation *maps* is given in (6.2).

Definition 6.8. Given an alphabet Σ, function $f : \Sigma^* \to \Sigma^*$, and word $w \in \Sigma^*$, the **mutation map** of f relative to w is the the restriction of f to the mutation points of f.

$$\overrightarrow{\mu}(f, w) := \{(x, f(x)) : x \in \mu(f, w)\} \tag{6.1}$$

Given functions $f, g : \Sigma^* \to \Sigma^*$, the composition $f \circ g$ is **μ-conserving** iff for all $w \in \Sigma^*$,

$$\overrightarrow{\mu}(f \circ g, w) = \overrightarrow{\mu}(g, w) \cup \overrightarrow{\mu}(g, w) \tag{6.2}$$

Given an alphabet Σ, phonological maps are functions from Σ^* to Σ^* (i.e. strings over the alphabet Σ). In Definition 6.8, the mutation map of a function is always defined with respect to a particular string. Consider the data points from Palestinian Arabic in (95). In (95a), both /a/ and /k/ become pharyngealized. This means that over the input string /Ṭuub-ak/, the pairs (a, A) and (k, K) are both in the mutation map of this function, with respect to the word /Ṭuub-ak/. However, in (95b) the process is blocked by the vowel /i/ and these sounds do not get pharyngealized, so (a, A) and (k, K) are *not* in the mutation map of the function with respect to the word /Ṭiin-ak/. It is for this reason that the definition of mutation maps of a string function $f : \Sigma^* \to \Sigma^*$ must be relativized to particular words $w \in \Sigma^*$.

(95) a. /Ṭuub-ak/ $\mapsto$ [ṬUUB-AK]
b. /Ṭiin-ak/ $\mapsto$ [Ṭiin-ak]
c. /Ṣayyad/ $\mapsto$ [ṢAyyad]
d. /ʔaṬfall/ $\mapsto$ [ʔAṬFALL]

In order to define weakly deterministic maps in terms of mutation maps, two further points need to be considered. First, note that a symbol can appear more than once in a single word. In (95a), (u, U) is in the mutation map of the function, but this change takes place twice in the word. The equation in (6.1) does not distinguish between two separate changes. In other words, (u, U) is in the mutation map of pharyngeal harmony with respect to the word /Ṭuub-ak/, without any distinction between the (u, U) change at index 1 and the (u, U) change at index 2. This is an important point because in (95c) the input string has two occurrences of /a/, one of which

undergoes pharyngealization and one which does not. The first /a/ in the input string become pharyngealized because of the emphatic sound /Ṣ/ immediately preceding it, but the second one does not because the harmony process gets blocked by /y/. Thus, it would not be sufficient to say (a, A) is in the mutation map with respect to the word /Ṣayyad/; we need to make specific reference to *which* /a/ in the string is in the mutation map.

The second point is that string functions in general do not have an a priori correspondence between elements of an input-output pairs of strings. In the input-output pair (Ṭuubak, ṬUUBAK), it seems evident that the first element of the input string maps to the second element of the output string, and similarly for each subsequential element. However, such a correspondence is not inherent to a string function and needs to be formally-defined. Consider again the LHOL stress map from (80), repeated below. Recall that this function can be modeled as the composition of a left subsequential (LS) and right subsequential function (RS), where LS stresses the leftmost heavy syllable if there is one, as in (80a), and a RS stresses the leftmost syllable if there are no heavy ones, as in (80b).

(80) *LHOL stress, as a map over* $\{H, L, \acute{H}, \acute{L}\}^*$

a. $\mathrm{L}^n\mathrm{H}w \mapsto \mathrm{L}^n\acute{\mathrm{H}}w$ for $n \geq 0$ and $w \in \{H, L\}^*$ (*Stress Leftmost heavy syllable*)

b. $\mathrm{L}^n \mapsto \acute{\mathrm{L}}\mathrm{L}^{n-1}$ for $n > 0$ (*Default to leftmost if no heavy syllables*)

The transducer which computes RS is given in Figure 6.10. This transducer reads strings from right to left. If it reads an L, it outputs the empty string λ because it does not know yet whether it will read an H or not. If the transducer reads an H input, then it goes to the state q_2 which outputs everything faithfully. If it gets to the end of the string without having read any H inputs, then it outputs a final stressed Ĺ at state q_1. In this way, the transducer stresses the final (leftmost) L if no H inputs are found in the string, and does not stress anything otherwise.

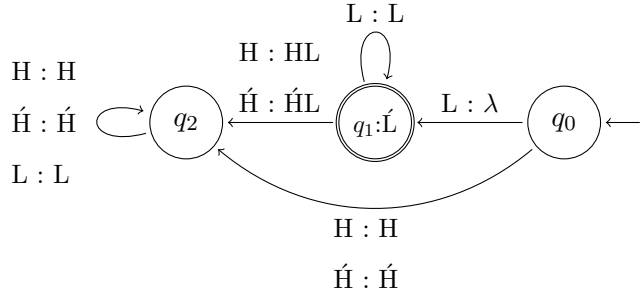


Figure 6.10: Right subsequential automata for LHOL stress, which computes the function in (80b) [Koser and Jardine, 2020].

Figure 6.11 shows three input-output pairs that this transducer produces. In 6.11(a), the automata reads the first H, outputs H, and goes to q_2 where all other inputs will also be output faithfully. This input-output pair therefore does not have any mutation points since no changes are made. The input-output pair in (b) also does not involve any change; 'LHLL' outputs as 'LHLL' because of the presence of a heavy syllable. Therefore this map should also not have any mutation points. In Figure 6.11(b) both the H syllable and rightmost L are in the set of mutation points of the function due to the fact that the automata has to wait after seeing the first L input. When an H input is read in the string, the transducer then outputs an unstressed L followed by an H. While the final output string is identical to the input string, it is shifted over in such a way that input-output correspondences in the transducer are not lined up one-to-one. A similar observation holds for the input 'LLLL' in (c). The transducer has to wait and output a stressed Ĺ only when it reaches the end of the string, at which point it knows that there are no H inputs in the string. Due to the fact that the transducer has to wait and shift everything over, the leftmost input L that gets stressed is technically not a mutation point, and the Ĺ output does not have a corresponding input.

input:	L	H	L	H
output:	L	H	L	H

(a) LHLH $\mapsto$ LHLH

input:	L	H	L	L
output:	L	HL	L	λ

(b) LHLL $\mapsto$ LHLL

input:		L	L	L	L
output:	Ĺ	L	L	L	λ

(c) LLLL $\mapsto$ ĹLLL

Figure 6.11: Examples of computations of right subsequential LHOL transducer

Thus, the mutation map of a function depends on how the input-output elements are aligned. The examples in Figure 6.11 show that the mutation map of a function may not capture the actual changes a function induces when the input-output correspondences are modeled by a transducer. For this reason, we further need origin semantics to specify the input-output correspondences between elements in a string [Bojańczyk, 2014, Bojańczyk et al., 2017, Dolatian et al., 2021b]. Meinhardt et al. [2021, pg.28] explain: "A transducer τ with origin semantics applied to an input string $w = x_1 \dots x_j \dots x_n$ not only returns an output string $\tau(w) = y_1 \dots y_i \dots y_n$ but is also associated with an *origin function* o that maps information about each position j in the output to information about what position $o(i) = j$ was rewritten in the input to produce the output symbol y_i at that output position."

To summarize, in order to describe the changes made by a string-to-string function we need to specify the changes made over particular strings, the position in the string where the change occurs, and a correspondence between the elements of the input output strings. All these properties of a function are represented by 'mutation factors' (Definition 6.9). Using this concept, Meinhardt et al. [2021] propose a definition of weakly deterministic functions presented in Definition 6.10.

Definition 6.9 (Mutation Factors)**.** Given an alphabet Σ, a string function $f : \Sigma^* \to \Sigma^*$, and word $w \in \Sigma^*$ the μ-factors f relative to w are

$$\overrightarrow{\mu}(f, w) = \{(o(i), x_{o_i}, i, y_i) \mid y_i \neq x_{\rtimes_i} \vee i \neq o(i)\} \tag{6.3}$$

Definition 6.10 (**Proposal 2: μ-Conserving**)**.** [Meinhardt, Mai, Baković, and McCollum, 2021] A regular function $\tau : X^* \to Z^*$ is weakly deterministic iff there exist contradirectional subsequential functions $I : X^* \to Y^*$ and $O : Y^* \to Z^*$ such that that following hold:

(a) $X \subseteq Y$,

(b) I and O are length-preserving and definable by transducers that are subsequential, synchronous, and equipped with origin semantics, and

(c) $O \circ I$ is μ-conserving: for all $w \in X^*$, $\overrightarrow{\mu}(O \circ I, w) = \overrightarrow{\mu}(O, w) \cup \overrightarrow{\mu}(I, w)$

Condition (iii) of Definition 6.10 formalizes the intuition that weakly deterministic functions are compositions of left and right subsequential function such that any change made to any element of input string is either a change made by the left subsequential function or the right subsequential function. Where the issue lies for this characterization of weakly deterministic functions in Definition 6.10 is in its complexity. To show that a function is *not* weakly deterministic, it is necessary to show that there do *not exist two functions* O and I which satisfy the conditions in (i)–(iii). For any such function, a proof would require showing that no left and right subsequential functions that are definable with transducers, and no origin semantics that they can be equipped with, can be composed to yield the function in such a way where the condition (iii) is satisfied. Due to the complex nature of the definition and all the individual components, it is unclear where this definition can be successfully applied to show that a pattern is not weakly deterministic. Ultimately, there is good intuitive reason to believe that Sour Grapes, Copperbelt Bemba high tone spreading,

and Tutrugbu ATR harmony cannot be described in terms of μ-conserving functions, but there no formal proof to go with the intuition. Thus, while Proposal 2 provides significant contributions to clarifying the difference between unbounded circumambient and weakly deterministic functions, we do not have evidence that this definition succeeds in separating the weakly deterministic and unbounded circumambient patterns.

The two proposal presented here are summarized in Table 6.12. The first column summarizes the empirical background in Section 6.1 and the intended characterizations of the weakly deterministic and unbounded circumambient patterns that were presented in Figure 6.1. Where the Heinz & Lai definition deviates from the first column is in its characterization of the Yidiny, Copperbelt Bemba, and Tutrugbu patterns as weakly deterministic. On the other hand, it is unclear whether the Meinhardt et. al. definition excludes these patterns. There is also a '?' for LHOL stress because of the above discussion about string functions and mutation maps. This pattern most likely does satisfy Definition 6.10 and an origin function which lines up the input and output elements as $o(i) = i$ may be sufficient in doing so. The final column of table is a summary of the results in Section 5, and shows how Definition 6.2 succeeds in matching the intended characterizations of the different maps presented in this paper.

	WD	Heinz & Lai	Meinhardt et. al.	Def 6.2
Palestinian Arabic Pharyngeal Harmony	yes	yes	yes	yes
LHOL Stress	yes	yes	?	yes
Yidiny Liquid Dissimilation	no	yes	?	no
Copperbelt Bemba Tone Spreading	no	yes	?	no
Tutrugbu ATR Harmony	no	yes	?	no
Sour Grapes Harmony	no	yes	?	no

Figure 6.12: Intended characterizations of various maps as weakly deterministic maps, compared to definitions proposed by Heinz and Lai [2013] (Definition 6.7), Meinhardt et al. [2021] (Definition 6.10), and this chapter (Definition 6.2)

The definition provided in this paper ultimately does succeed in including the weakly deterministic maps discussed in Section 6.1, while also successfully *excluding* the non-weakly deterministic ones. However, the motivation and intuition behind Definition 6.2 is built off of discussion from Heinz and Lai [2013] and Meinhardt et al. [2021]. In Heinz & Lai's definition, the purpose of using a constraint against mark-up as the characteristic property of weakly deterministic functions is to ensure that one subsequential function cannot pass information to another through an intermediate

representation, as in the example of high tone spreading in Figure 6.8. Definition 6.2 shares exactly this intuition; the simultaneous application of two functions is a single function applied over the input. Because there is no intermediate representation, no kind of mark-up can be used. The phonotactic codes used by McCollum et al. [2020] and Lamont [2019] are therefore not possible because both subsequential functions in the composition apply directly to the input. Thus the motivation for definition weakly deterministic functions in terms of simultaneous application is to ensure that mark-up is not possible by eliminating an intermediate representation.

With respect to the Meinhardt et. al. definition, the purpose of mutation maps is to capture the fact that a weakly deterministic function is one in which every segment in the input only needs information either to the right or to the left, and can therefore be explained in terms of either a right subsequential *or* a left subsequential function. To see how the simultaneous application operator defined in this paper captures the same idea as mutation conserving compositions in Meinhardt et al. [2021], consider again the map /ʔaṮfall/ ↦ [ʔAṬFALL] in (95d), where emphasis spread takes place both to the left and right of the trigger /Ṭ/. The first two elements in the input string become pharyngealized because of a trigger to the *right* while the final four elements become pharyngealized because of a trigger to the *left*. Figures 6.3 and 6.5(d) are repeated below in Figure 6.13 to show how simultaneous application and μ-consreving compositions capture this idea. Figure 6.13(a) shows how the pattern can be expressed as the simultaneous application of RTR^R and RTR^L such that the first two segment becomes pharyngealized because of RTR^R while the final four become pharyngealized because of RTR^L. Figure 6.13(b) shows how all the changes in the input-output mapping can be expressed as changes made by a right subsequential function RS and a left subsequential function LS, and therefore the mutation map of the entire function is the union of the mutation maps of LS and RS.

BMRS makes it possible to express exactly the same ideas and intuitions as Definition 6.10 in a much simpler way. First, being a left or right subsequential function is characterized in terms of programs using predecessor or successor. We therefore only need to reason about what kind of information is accessible through the use of the predecessor and successor functions. Second, the correspondences between input and output segments is built directly into the BMRS framework because the domain of a program is a set of *indices*. This allows us to refer to the input and output properties at a particular index. In a way, there is an implicit origin map built into the way

input	ʔ	a	ṭ	f	a	l	l
$\mathrm{RTR}(x)$	⊥	⊥	⊤	⊥	⊥	⊥	⊥
$\mathrm{RTR}^R(x)$	⊤	⊤	⊤	⊥	⊥	⊥	⊥
$\mathrm{RTR}^L(x)$	⊥	⊥	⊤	⊤	⊤	⊤	⊤
$\mathrm{RTR}^{R \otimes L}(x)$	⊤	⊤	⊤	⊤	⊤	⊤	⊤
output	ʔ	A	ṭ	F	A	L	L

(a) Simultaneous application of RTR^R and RTR^L

input	ʔ	a	ṭ	f	a	l	l
output	ʔ	A	ṭ	F	A	L	L
	$\overrightarrow{\mu}(RS)$			$\overrightarrow{\mu}(LS)$			

(b) Mutation map of $RS \circ LS$

Figure 6.13: Conceptual equivalence of simultaneous application of BMRS programs in (a) and μ-conserving compositions in (b) over the example of bidirectional emphasis spreading in Palestinian Arabic.

programs are defined. Third, the definition of simultaneous application is a logical one, defined with Boolean values. We therefore only need to reason about when Boolean values get flipped (as in Theorem 6.3). These three properties of BMRS make reasoning about what is and is not weakly deterministic much simpler.

6.4 Discussion

This chapter showed how the model-theoretic framework of BMRS makes it possible to reason about the complexity of phonological maps. Chapter 3 of this dissertation discussed the logical characterization of the ISL, subsequential, and rational maps in terms of BMRS transductions. This chapter showed that it is not only possible to give a *logical* characterization of weak determinism in terms of BMRS transductions, but that doing so also makes it possible to reason about what is and is not weakly deterministic. Ultimately, this is what distinguishes the logical method in this chapter from previous attempts at characterizing weak determinism.

More recently, Meinhardt et al. [2024] gave an automata-theoretic characterization of weak determinism in terms of bimachines. Bimachines are transducers that are constructed from two automata, one which reads a string left-to-right, and another which reads the string right-to-left. These transducers compute regular string functions [Schützenberger, 1961, Mihov and Schulz, 2019]. Meinhardt et al. [2024]'s characterization of weak determinism as a constraint on bimachines is built

off the same intuition discussed in Meinhardt et al. [2021] and this chapter. However, their paper does not provide proof that various unbounded circumambient maps are *not* weakly deterministic with respect to their proposed characterization. Two open questions left to explore in future work is whether their definition succeeds in separating the weakly deterministic and unbounded circumambient maps, and whether their definition is equivalent to Definition 6.2.

A further avenue of future work is the logical characterization of unbounded circumambient maps. In terms of logical description, the difference between weakly deterministic and unbounded circumambient patterns can be described with disjunction and conjunction. The weakly deterministic functions are are such that they need information either to the left *or* to the right while the unbounded circumambient ones are such that they need information to the left *and* to the right. The simultaneous application operator in Chapter 5 is disjunctive; the simultaneous application of two predicates/programs changes a segment in the input if and only if either one of them does. This property made it well-suited for characterizing the weakly deterministic maps. It is possible to define a *conjunctive* form of simultaneous application, which changes a segment in the input if and only if *both* P^f *and* P^g change it. This operator, denoted $\mathbin{⍉}$, is described in Figure 6.14 and formalized over BMRS expressions in Definition 6.11.

$P(x)$	$P^f(x)$	$P^g(x)$	$P^{f \mathbin{⍉} g}(x)$
$\top$	$\top$	$\top$	$\top$
$\top$	$\top$	$\bot$	$\top$
$\top$	$\bot$	$\top$	$\top$
$\top$	$\bot$	$\bot$	$\bot$
$\bot$	$\top$	$\top$	$\top$
$\bot$	$\top$	$\bot$	$\bot$
$\bot$	$\bot$	$\top$	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$

Figure 6.14: Conjunctive simultaneous application of two output predicates, as a truth table (originally presented as Figure 6.14 in Section 5.4). If both $P^f(x)$ and $P^g(x)$ flip the truth value associated with $P(x)$, then $P^{f \mathbin{⍉} g}(x)$ flips the truth value (highlighted).

Definition 6.11 (Conjunctive Simultaneous Application)**.** For an input predicate P and two output predicates P^f and P^g, the conjunctive simultaneous application of P^f and P^g is defined as

$$P^{f \mathbin{⍉} g}(x) := \text{if } P(x) \text{ then } \big(P^f(x) \vee P^g(x)\big) \text{ else } \big(P^f(x) \wedge P^g(x)\big) \tag{6.4}$$

The difference between Definitions 5.4 (simultaneous application) and 6.11 (conjunctive simul-

taneous application) is the swapping of conjunction and disjunction in the BMRS expression. In the case of *conjunctive simultaneous application*, if a property/feature P holds of some x in the input, then it will also be true of x in the output if and only if either P^f or P^g *does not* flip the truth value. In other words, either P^f or P^g must evaluate to $\top$. In this case, the disjunction $P^f(x) \vee P^g(x)$ must hold. If P does not hold of some x in the input, then P will hold of x in the output if and only if both P^f and P^g flip the truth value. In this case, the conjunction $P^f(x) \wedge P^g(x)$ must hold.

Recall that Sour Grapes harmony cannot be modeled as the simultaneous application of two programs where one uses only predecessor and the other uses only successors (Theorem 6.3). However, it *can* be modeled as the simultaneous application of one expression which only uses predecessor and another which only uses successor. For the harmonic feature F, the two expressions F^L and F^R which accomplish this are presented in (96). The predicate P^R requires an auxiliary function NoB-R(x), which evaluates to $\top$ if and only if there is no blocker anywhere to the right of x. This function is defined exactly like NoH-R in (81b). Figure 6.15 presents the operators $\oslash$ and $\oslash$ over the expressions F^L and F^R. These tables illustrate how $F^{L \oslash R}$ yields Sour Grapes harmony.

(96) *Sour Grapes is the conjunctive simultaneous application of $F^L \in BMRS^p$ and $F^R \in BMRS^s$*

$F^L(x)$ = if $F(x)$ then $\top$ else $F(p(x))$

$F^R(x)$ = if $F(x)$ then $\top$ else $\big($if $B(x)$ then $\bot$ else NoB-R$(x)\big)$

	+	−	−	⊟	−	−
$F(x)$	⊤	⊥	⊥	⊥	⊥	⊥
$B(x)$				⊤		
NoB-R(x)	⊥	⊥	⊥	⊤	⊤	⊤
$F^L(x)$	⊤	⊤	⊤	⊥	⊥	⊥
$F^R(x)$	⊤	⊥	⊥	⊥	⊤	⊤
$F^{L \oslash R}(x)$	⊤	⊤	⊤	⊥	⊤	⊤
$F^{L \oslash R}(x)$	⊤	⊥	⊥	⊥	⊥	⊥
$L \oslash R$	+	+	+	⊟	+	+
$L \oslash R$	+	−	−	⊟	−	−

(a)

	+	−	−	−
$F(x)$	⊤	⊥	⊥	⊥
$F^L(x)$	⊤	⊤	⊤	⊤
$F^R(x)$	⊤	⊤	⊤	⊤
$F^{f \oslash g}(x)$	⊤	⊤	⊤	⊤
$L \oslash R$	+	+	+	+

(b)

Figure 6.15: Sour Grapes as the conjunctive simultaneous application of $F^L \in$BMRS^p and $F^R \in$BMRS^r, where F is the harmonic feature, B is the blocker, and NoB-R$(x) = \top$ iff there is no blocker to the right of x.

Consider first Figure 6.15(a). The second-to-last row of the table shows how $F^{L \oslash R}$ yields the transformation $+ - - \boxminus - - \;\mapsto\; + + + \boxminus + +$. The simultaneous application of F^L and F^R

essentially ensures that spreading of the feature F will take place if there is a segment with feature F on the left *or* there is no blocker on the right. The conjunctive form of simultaneous application, on the other hand, ensures that spreading of the feature F will take place if there is something with feature F on the left *and* no blocker on the right. The last rows of the two tables in Figure 6.15 show how Sour Grapes harmony can be decomposed into a left and right subsequential function via a *conjunctive* simultaneous application operator.

The class of unbounded circumambient maps does not have a logical characterization. The example of Sour Grapes shows that the conjunctive form of simultaneous application may provide such a characterization. The potential value of this work would be that it formalizes the boundary between weakly deterministic and unbounded circumambient patterns in terms of logic. Such a formalization would have important consequences for how the space of regular functions is partitioned. In particular, are all rational functions either weakly deterministic or unbounded circumambient, or are there rational functions which are neither? A logical characterization of unbounded circumambient patterns within BMRS may make it possible to answer this question.

7 DISCUSSION AND FUTURE WORK

This dissertation provided an in-depth introduction to the BMRS framework as a formal tool for modeling phonological maps as logical transductions, and discussed the advantages of logical representations. This chapter discusses some promising avenues of future research which arise out of this project.

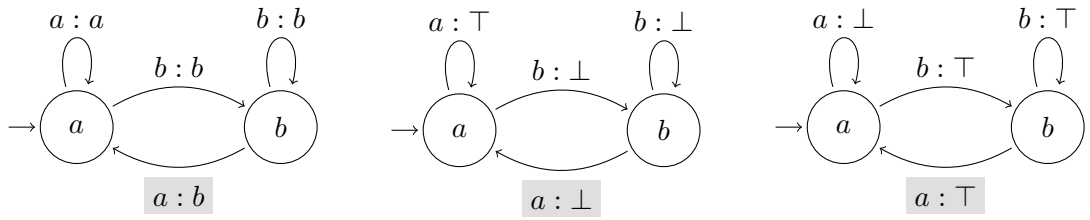
7.1 Programs and Automata Revisted

Section 3.2.1 showed how finite state automata can be represented with equivalent logical transducers which simulate it. This section presents another way to think about the relationship between logical transducers and FSTs which highlights an interesting parallel between the representations and learning procedure presented here and the representations and procedure proposed by Markowska and Heinz [2023]. In particular, each line of a program can be equivalently expressed as an FST which outputs a Boolean value rather than an element of the alphabet. To demonstrate this, consider again the simple string function expressed by the rewrite rule $a \to b / b\_\_$. The BMRS program which represents this string function is repeated below in (6). The FST which represents this string function is presented in Figure 7.1(a).

(6) *The function $a \to b / b\_\_$ as a BMRS program*

$$P'_a(x) = \text{if } P_b(p(x)) \text{ then } \bot \text{ else } P_a(x)$$
$$P'_b(x) = \text{if } P_a(x) \text{ then } P_b(p(x)) \text{ else } P_b(x)$$

Each individual predicate $P'_\sigma(x)$ can also be represented as an FST with an input alphabet $\{a, b\}$



(a) $\{P_a^f(x), P_b^f(x)\}$ as an FST. (b) $P_a^f(x)$ as an FST. (c) $P_b^f(x)$ as an FST

Figure 7.1: FST representation of the string function expressed by the program in (6), and its factorization into two FSTs, one for each line of the program.

and output alphabet $\{\top, \bot\}$ such that the transducer outputs $\top$ after reading the symbol at index x of the input string if and only if $P_\sigma(x) = \top$. For an alphabet of n symbols, we get a factorization of the function into n separate FSTs, one for each predicate. The associated factorization of the string function in (6) are presented in Figure 7.1(b,c). The relationship between the transducers in 7.1(a,b,c) and the BMRS program in (6) is illustrated in the table in Figure 7.2. The string 'abbba' is the result of running the input string 'abbaa' through the transducer in Figure 7.1(a). It is also the solution to the system of equations $\{P'_a(x), P'_b(x)\}$ in (6) applied over the string model $\mathcal{M}(abbaa)$. The truth values in the table are the truth values corresponding with each of the predicates $\{P'_a(x), P'_b(x)\}$ at the indices $\{0, \ldots, 5\}$. The row of truth values associated with the predicate $P'_a(x)$ is exactly the output of the FST in Figure 7.1(b) over the input string 'abbaa'. Similarly, the row of truth values associated with the predicate $P'_b(x)$ is the output of the FST in Figure 7.1(c) over the same input string. In this way, the BMRS program in (6), the FST in Figure 7.1(a), and the pair of FSTs in Figure 7.1(b,c) are all equivalent. They are three different ways of representing the same string function.

	a	b	b	a	a
	1	2	3	4	5
$P_a^f(x)$	$\top$	$\bot$	$\bot$	$\bot$	$\top$
$P_b^f(x)$	$\bot$	$\top$	$\top$	$\top$	$\bot$
	a	b	b	b	a

Figure 7.2: The program in (6) over the input string 'abbaa'. The transducer in Figure 7.1(a) corresponds with the input-output pair of strings. The transducers in Figures 7.1(b,c) correspond with each of the rows of the table.

Programs over feature representations have n equations for an inventory of n features, and a factorization of a phonological map into n FSTs. For simplicity, we consider the vowel nasalization

process in (97) which only makes use of two features: [cons] and [nas]. In order to keep the transducers small, we further assume that [-cons,+nas] sounds never appear in the input. There are therefore three states: $--$, $++$, and $+-$, corresponding with each of the possible feature values for [cons] and [nas]. The state $--$ for example, corresponds with a [-cons,-nas] input. Figure 7.3 presents the FST for vowel nasalization. Similar to the previous example, each line of the program in (97) can be expressed with an FST that computes that line, presented in Figure 7.4. The transducer in Figure 7.4(a) represents the equation $[cons]'(x)$ and the transducer in (b) represents the equation $[nas]'(x)$. More specifically, the transducer in (b) outputs $\top$ for the input value at index x of a string w exactly when $[nas]'$ returns $\top$ for index x of the string model $\mathcal{M}(w)$. Similar to the previous example, the BMRS program in (97), the FST in Figure 7.3, and the pair of FSTs in Figure 7.4 all represent the same phonological map.

(97) *Vowel nasalization:* [-cons]→[+nas]/[+cons,+nas]__

$$
\begin{aligned}
[cons]'(x) &= [cons](x) \\
[nas]'(x) &= \text{if } [nas](x) \text{ then } \top \text{ else} \\
&\quad \text{if } [cons]'(x) \text{ then } [nas](x) \text{ else} \\
&\quad [nas](px)
\end{aligned}
$$

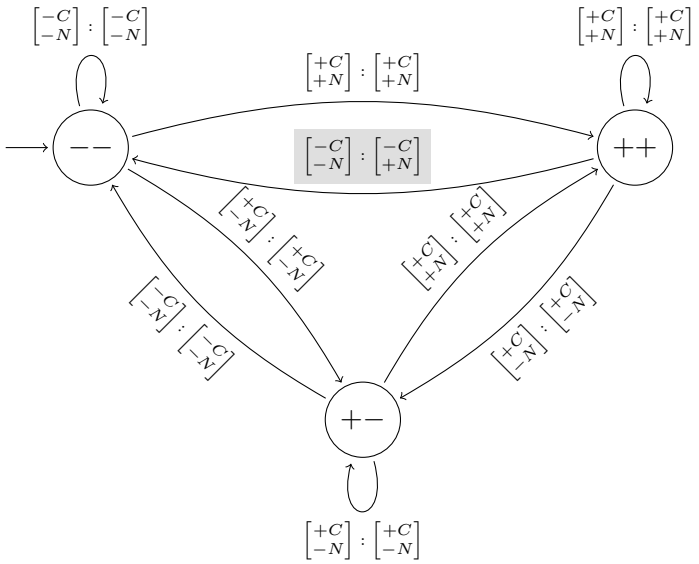
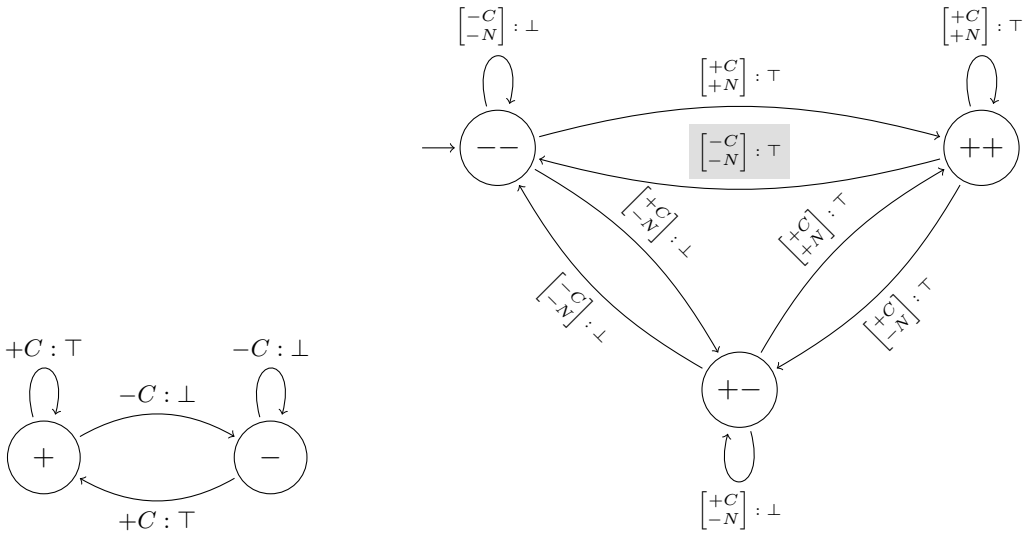


Figure 7.3: FST which computes the vowel nasalization map in (97).

The significance of the factorized transducers in Figure 7.4 is that they are reminiscent of the learning algorithm proposed by Markowska and Heinz [2023]. They argue that using properties



(a) FST representation of $[cont]'(x)$ (b) FST representation of $[nas]'(x)$

Figure 7.4: Factorization of program in (97) into two FSTs, one for each line of the program.

of letters in the alphabet (i.e. features) rather than the alphabet symbols reduces the size of the alphabet because n features can be used to represent 2^n symbols. Their learning approach proposes a factorization of a subsequential function into n-many subsequential functions (one for each binary feature), to be represented as deterministic FSTs. Each FST in the factorization can then be learned via the Structured Onward Sequential Function Inference Algorithm (SOSFIA) which was introduced by Jardine et al. [2014]. A BMRS representation of phonological maps with features amounts to the same idea: a factorization of a map into separate transductions, one for each feature. Thus, the underlying idea behind this work and Markowska and Heinz [2023] comes down to the same higher level idea of breaking a phonological map down into several smaller representations, and adapting a previously-developed learning algorithm to each of the smaller problems simultaneously in order to learn the larger map. In the case of Markowska and Heinz [2023], the problem is broken down into smaller FSTs, where SOSFIA can be adapted to learning each individual FST. In this case of Chapter 4 of this dissertation, the problem is broken down into learning each line of the BMRS program where BUFIA can adapted to learning each individual line simultaneously. The equivalence between the logical transducers and factorized FSTs, however, shows that the representation of phonological maps between the two approaches is equivalent. An interesting area of research that remains to be explored is an experimental study on these two

learning procedures. A more precise question to answer is: Are there advantages to the model-theoretic representations used in this dissertation over FSTs? Markowska and Heinz provide an excellent place to approach this question from the perspective of learning.

7.2 Representation and Locality

This dissertation only discussed string representations of words. Moreover, the logical characterizations from Bhaskar et al. [2020, 2023] and Chandlee and Lindell [forth.] are based on string representations. However, Chapter 3 alluded to the fact that model-theoretic structures can be enriched with relations/functions that represent tiers, autosegmental structure, syllables, and prosodic structure. Tiers, for example, are represented as an ordering on a subset of the alphabet. Figure 2.1 in Section 2.1 presented the successor model $\mathcal{M}^{\triangleleft}(baa)$. This model can be enhanced with a tier-successor relation $\triangleleft^a$, which is the successor relation restricted to the alphabet $\{a\}$ [Lambert and Rogers, 2020]. The tier-successor model for the string 'baa' is presented in Figure 7.5.

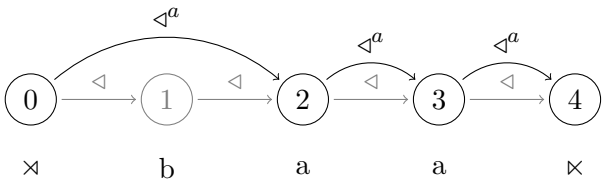


Figure 7.5: String model $\mathcal{M}(baa)$ enhanced with a tier-successor relation on $\{a\}$ [Lambert and Rogers, 2020].

The significance of exploring more complex representations of words is that many phonological patterns reduce to local rules/processes over these enriched representations [Lamont, 2023, Graf, 2023, Blum, 2023, Koser, 2022, Burness, 2022, Burness et al., 2021, Jardine, 2019, 2017, 2016b, Aksënova et al., 2016, McMullin and Hansson, 2016]. Recall the liquid dissimilation map from Georgian, where the /r/ sound in the /-uri/ suffix undergoes dissimilation when the stem has a /r/ sound, as long as there is no intervening /l/. This function was presented (17) of Section 3.1.1 as an example of a left subsequential map. This pattern can be expressed with a surface constraint ${}^*r < r$ over predecessor models, or a *local* constraint ${}^*r \triangleleft^{\{l,r\}} r$ over tier models [Jardine and Heinz, 2016, Heinz et al., 2011]. The word /ast'**r**onomi-u**r**i/ 'astronomical', for example, violates ${}^*r \triangleleft^{\{l,r\}} r$ while the word /ka**rtl**-u**r**i/ 'Kartvelian' does not. The significance of the liquid dissimilation process is that a map which requires recursion over string representations, *does not* require recursion over tier representations. A similar observation holds of the Tutrugbu ATR harmony pattern in (70)

of Section 3.1.2, which McCollum et al. [2020] argues is unbounded circumambient. Blum [2023], however, argues that over multi-tiered autosegmental representations, Tutrugbu ATR harmony reduces to a strictly local pattern. A similar result holds of stress patterns; Koser [2022] shows that enhancing representations with metrical feet reduces long-distance (subsequential) maps to a combination of OSL and ISL maps. The relationship between enriched representations and locality moreover goes beyond the domain of phonology. Aksënova et al. [2016] show that many morphotactic phenomena are tier-based strictly local. Within syntax, the tier-based strictly local class has also been proposed as an upper bound on the complexity of many movement Graf [2023] and agreement patterns Hanson [2025].

With respect to phonological *maps*, the tier-based strictly local functions [Burness and McMullin, 2020, 2019, Hao and Andersson, 2019] extend the concept of ISL and OSL maps to function that operate over tiers. One question that this paper did not explore is whether the maps in Sections 3.1 and 6.1.1 reduce to local patterns over more complex representations such as tiers. In other words, are any of the subsequential maps tier-based strictly local? Which of the unbounded circumambient patterns reduce to strictly local over a more-enriched representation? Moreover, weakly deterministic maps were characterized as the simultaneous application of contradirectional subsequential maps in Chapter 6. Can we say something stronger? That is, can weakly deterministic maps instead be expressed as the simultaneous application of tier-based left and right output strictly local maps instead? An interesting pursuit for future work is re-examining the complexity of phonological maps with more complex model-theoretic representations.

The question of whether maps outside the ISL class reduce to local maps over enriched representations is of particular importance to the learning procedure presented in Chapter 4. While the chapter only discussed non-recursive programs, it is easy to conceive how we could extend the hypothesis space to also encode local information about the output string. However, functions outside the ISL and OSL class are a challenge. Recall the string function f^* expressed with the rewrite rule $a \rightarrow b / bx^*\_\_$; an 'a' changes to a 'b' if there is a 'b' anywhere to the left of it in the string. Section 2.2.2 discussed how the BMRS program which expresses this function requires an auxiliary recursive predicate `b-left` which returns $\top$ for an index x iff there is a 'b' in some index $y < x$. The hypothesis space cannot be extended to include auxiliary predicates such as `b-left` because there are infinitely many such predicates that can be defined. However, if these

long-distance processes reduce to local processes over non-linear representations, then they are no longer an obstacle. The solution in that case would be the extend the hypothesis space to include more complex representations. Thus a study of the complexity of phonological maps over various representations has significant implications for learnability.

BIBLIOGRAPHY

Stan Abbott. A tentative multilevel multiunit phonological analysis of the Murik language. *Papers in New Guinea Linguistics*, 22:339–373, 1985.

Alëna Aksënova, Thomas Graf, and Sedigheh Moradi. Morphotactics as tier-based strictly local dependencies. In Micha Elsner and Sandra Kuebler, editors, *Proceedings of the 14th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 121–130. Association for Computational Linguistics, 2016. doi: 10.18653/v1/W16-2019.

Richard Applegate. *Ineseño chumash grammar*. PhD thesis, University of California, Berkeley, 1972.

Eric Baković and Lev Blumenfeld. A formal typology of process interactions. *Phonological Data and Analysis*, 6(3):1–43, 2024. doi: 10.3765/pda.v6art3.83.

Eric Baković. *Harmony, Dominance and Control*. PhD thesis, Rutgers University, 2000.

Eric Baković. Vowel harmony and stem identity. *Rutgers Optimality Archive*, (540), 2003.

Eric Baković. *Opacity and Ordering*, chapter 2, pages 40–67. Wiley Online Library, 2011. doi: 10.1002/9781444343069.ch2.

Kenneth R. Beesley and Lauri Karttunen. *Finite-State Morphology*. CSLI Publications, 2003.

William Bennet. *Dissimilation, consonant harmony, and surface correspondence*. PhD thesis, Rutgers University, 2013.

Siddharth Bhaskar, Jane Chandlee, Adam Jardine, and Christopher Oakden. Boolean monadic recursive schemes as a logical characterization of the subsequential functions. In Alberto Leporati, Carlos Martín-Vide, Dana Shapira, and Claudio Zandron, editors, *Language and Automata Theory and Applications (LATA)*, Lecture Notes in Computer Science, pages 157–169. Springer, 2020. doi: 10.1007/978-3-030-40608-0_10.

Siddharth Bhaskar, Jane Chandlee, and Adam Jardine. Rational functions via recursive schemes, 2023.

Lee S Bickmore and Nancy C Kula. Ternary spreading and the OCP in copperbelt bemba. *Studies in African Linguistics*, 42(2):101–132, 2013. doi: 10.32473/sal.v42i2.107270.

Eileen Blum. *The effects of non-linear data structures on the computation of vowel harmony*. PhD thesis, Rutgers University, New Brunswick, 2023.

MikołBojańczyk. Transducers with origin information. In *International Colloquium on Autamata, Languages, and Programming*. Springer, 2014.

MikołBojańczyk, Laure Daviaud, Bruno Guillon, and Vincent Penelle. Which classes of origin graphs are generated by transducers. In *International Colloquium on Automata, Languages, and Programming*, 2017.

Geert Booij. *The Phonology of*. Oxford Academic Press, 1999.

Richard Buchi. Weak second-order arithmetic and finite automata. *Mathematical Logic Quarterly*, 6:66–92, 1960. doi: 10.1002/malq.19600060105.

Phillip Burness. *Non-local phonological processes as Multi-Tiered Strictly Local maps*. PhD thesis, University of Ottawa, 2022.

Phillip Burness and Kevin McMullin. Efficient learning of output tier-based strictly 2-local functions. In *Proceedings of the 16th Meeting on the Mathematics of Language*, pages 78–90. Association for Computational Linguistics, 2019. doi: 10.18653/v1/W19-5707.

Phillip Burness and Kevin McMullin. Multi-tiered strictly local functions. In *Proceedings of the 17th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 245–255. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.sigmorphon-1.29.

Phillip Burness, Kevin James McMullin, and Jane Chandlee. Long-distance phonological processes as tier-based strictly local functions. *Glossa: a journal of general linguistics*, 6(1), 2021. doi: 10.16995/glossa.5780.

Jane Chandlee. *Strictly Local Phonological Processes*. PhD thesis, University of Delaware, 2014.

Jane Chandlee and Jeffrey Heinz. Bounded copying is subsequential: Implications for metathesis and reduplication. In *Proceedings of the Twelfth Meeting of the Special Interest Group on Computational Morphology and Phonology*, pages 42–51, 2012.

Jane Chandlee and Jeffrey Heinz. Strict locality and phonological maps. *Linguistic Inquiry*, 49(1):23–60, 2018. doi: 10.1162/LING_a_00265.

Jane Chandlee and Adam Jardine. Quantifier-free least fixed point functions for phonology. In *Proceedings of the 16th Meeting on the Mathematics of Language (MOL)*, pages 50–62. Association for Computational Linguistics, 2019a.

Jane Chandlee and Adam Jardine. Autosegmental input strictly local functions. In *Transactions of the Association for Computational Linguistics*, volume 7, 2019b. doi: 10.1162/tacl_a_00260.

Jane Chandlee and Adam Jardine. Computational universals in linguistic theory: Using recursive programs for phonological analysis. *Language*, 97:485–519, 2021. doi: 10.1353/lan.2021.0045.

Jane Chandlee and Adam Jardine. Phonological theory and computational modelling. In B. Elan Dresher and Harry van der Hulst, editors, *The Oxford History of Phonology*, chapter 31. Oxford University Press, 2022. doi: 10.1093/oso/9780198796800.003.0031.

Jane Chandlee and Steven Lindell. Logical perspectives on strictly local transformations. In *Doing Computational Phonology*. Oxford University Press, forth.

Jane Chandlee, Angeliki Athanasopoulou, and Jeffrey Heinz. Evidence for classifying metathesis patterns as subsequential. In *The Proceedings of the 29th West Coast Conference on Formal Linguistics*, pages 303–309, 2012.

Jane Chandlee, Rémi Eyraud, and Jeffrey Heinz. Learning strictly local subsequential functions. *Transactions of the Association for Computational Linguistics*, 2:491–504, 2014. doi: 10.1162/tacl_a_00198.

Jane Chandlee, Rémi Eyraud, and Jeffrey Heinz. Output strictly local functions. In *Proceedings of the 14th Meeting on the Mathematics of Language*, page 112–125. Association for Computational Linguistics, 2015. doi: 10.3115/v1/W15-2310.

Jane Chandlee, Remi Eyraud, Jeffrey Heinz, Adam Jardine, and Jonathan Rawski. Learning with partially ordered representations. In Philippe de Groote, Frank Drewes, and Gerald Penn, editors, *Proceedings of the 16th Meeting on the Mathematics of Language*, pages 91–101, Toronto, Canada, 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-5708.

Noam Chomsky. Three models for the description of language. *IRE Transactions on Information Theory*, 2(3):113–124, 1956.

Noam Chomsky. On certain formal properties of grammars. *Information and Control*, 2(2):137–167, 1959.

Noam Chomsky. *Aspects of the Theory of Syntax*. MIT Press, 1965.

Noam Chomsky and Morris Halle. *The Sound Patterns of English*. Harper & Row, 1968.

George Nick Clements. Akan vowel harmony: A nonlinear analysis. *Harvard Studies in Phonology*, 2, 1981.

George Nick Clements. Akan vowel harmony: A nonlinear analysis. In Didier L. Goyvaerts, editor, *African Linguistics: Essays in Memory of M.W.K. Semikenke*, volume 6 of *Studies in the Sciences of Language*. John Benjamins, 1985. doi: 10.1075/ssls.6.06cle.

Nick Clements and Elizabeth Hume. The internal organisation of speech sounds. In John Goldsmith, editor, *The Handbook of Phonological Theory*, chapter 7, pages 245–307. Blackwell, 1995.

Bruno Courcelle. Monadic second-order definable graph transductions: a survey. *Theoretical Computer Science*, 126(1):53–75, 1994. doi: 10.1016/0304-3975(94)90268-2.

Bruno Courcelle and Joost Engelfriet. Monadic second-order transductions. In *Graph Structure and Monadic Second-Order Logic: A Language-Theoretic Approach*, chapter 7, pages 505–577. Cambridge University Press, 2012. doi: 10.1017/CBO9780511977619.

Stuart Davis. Emphasis spread in Arabic and grounded phonology. *Linguistic Inquiry*, 26(3): 465–498, 1995.

Willa Dawson. *Tibetan Phonology*. PhD thesis, University of Washington, 1980.

Aniello De Santo and Jonathan Rawski. Mathematical linguistics and cognitive complexity. In *Handbook of Cognitive Mathematics*, pages 1–38. Springer, 2022.

Gerrit Jan Dimmendaal. *The Turkana Language*. De Gruyter Mouton, Berlin, Boston, 1983. doi: 10.1515/9783110869149.

Robert Dixon. *A grammar of Yidin*. Cambridge University Press, 1977.

Hossep Dolatian. *Computational locality of cyclic phonology in Armenian*. PhD thesis, Stony Brook University, 2020.

Hossep Dolatian, Nate Koser, Jonathan Rawski, and Kristina Strother-Garcia. Computational restrictions on iterative prosodic processes. In *Proceedings of the 2020 Annual Meeting on Phonology*, 2021a. doi: 10.3765/amp.v9i0.4920.

Hossep Dolatian, Jonathan Rawski, and Jeffrey Heinz. Strong generative capacity of morphological processes. In *Proceedings of the Society for Computation in Linguistics (SCiL)*, pages 31–42, 2021b.

Florence Abena Dolphyne. *The Akan (twi-fante) language : its sound systems and tonal structure*. Ghana Universities Press, 1988.

Calvin C Elgot and Jorge E Mezei. On relations defined by generalized finite automata. *IBM Journal of Research and Development*, 9(1):47–68, 1965.

Herbert Enderton. *A Mathematical Introduction to Logic*. Harcourt Academic Press, 1972.

Joost Engelfriet and Hendrik Jan Hoogeboom. Two-way finite state transducers and monadic second-order logic. In *26th International Colloquium on Automata, Languages and Programming*, 1999. doi: 10.1007/3-540-48523-6_28.

Joost Engelfriet and Hendrik Jan Hoogeboom. Mso definable string transductions and two-way finite-state transducers. *ACM Transactions on Computational Logic (TOCL)*, 2(2):216–254, 2001. doi: 10.1145/371316.371512.

James Essegbey. Noun classes in tutrugbu. *Journal of West African Languages*, 36:37–56, 2009.

James Essegbey. *Tutrugbu (Nyangbo) Language and Culture*. Brill, 2019.

Paul D. Fallon. Liquid dissimilation in Georgian. In *Eastern States Conference on Linguistics*, volume 10, pages 105–116, 1993.

Emmanuel Filiot. Logic-automata connections for transformations. In *Indian Conference on Logic and Its Applications*, page 30–57. Springer, 2015. doi: 10.1007/978-3-662-45824-2_3.

Emmanuel Filiot and Pierre-Alain Reynier. Transducers, logic and algebra for functions of finite words. *ACM SIGLOG News*, 3(3):4–19, 2016. doi: 10.1145/2984450.2984453.

Emmanuel Filiot, Olivier Gauwin, and Nathan Lhote. Logical and algebraic characterizations of rational transductions. *Logical Methods in Computer Science*, 15(4), 2019. doi: 10.23638/LMCS-15(4:16)2019.

Sara Finley. *The formal and cognitive restrictions on vowel harmony*. PhD thesis, Johns Hopkins University, 2008.

Sara Finley. The privileged status of locality in consonant harmony. *Journal of memory and language*, 65:74–83, 2011. doi: 10.1016/j.jml.2011.02.006.

Sara Finley. Testing the limits of long-distance learning: Learning beyond a three-segment window. *Cognitive Science*, 36:740—-756, 2012.

Sara Finley. Locality and harmony: Perspectives from artificial grammar learning. *Language and Linguistics Compass*, 11(1):1–16, 2017.

Sara Finley and William Badecker. Analytic biases for vowel harmony languages. In *Proceedings of the West Coast Conference on Formal Linguistics*, volume 27, pages 168–176, 2008.

Brian Gainor, Regine Lai, and Jeffrey Heinz. Computational characterizations of vowel harmony patterns and pathologies. In *The proceedings of the 29th West Coast Conference on Formal Linguistics*, pages 63–71, Somerville, MA, 2012. Cascadilla Press.

Daniel Gildea and Dan Jurafsky. Automatic induction of finite state transducers for simple phonological rules. In *Annual Meeting of the Association for Computational Linguistics*, 1995.

Matthew Gordon. *Syllable Weight: Phonetics, Phonology, Typology*. Routledge, 2006. doi: 10.4324/9780203944028.

Thomas Graf. Comparing incomparable frameworks: A model theoretic approach to phonology. *Proceedings of the Annual Penn Linguistics Colloquium*, 16, 2010.

Thomas Graf. Subregular tree transductions, movement, copies, traces, and the ban on improper movement. In *Proceedings of the Society for Computation in Linguistics*, page 289–299. Association for Computational Linguistics, 2023.

Kenneth Hanson. Tier-based strict locality and the typology of agreement. *Journal of Language Modelling*, 13(1):43–97, 2025. doi: 10.15398/jlm.v13i1.411.

Gunnar Ólafur Hansson. *Theoretical and typological issues in consonant harmony*. PhD thesis, University of California, Berkeley, 2001.

Gunnar Ólafur Hansson. *Consonant harmony: Long-distance interactions in phonology*, volume 145 of *UC Publications in Linguistics*. Univ of California Press, 2010.

Yiding Hao and Samuel Andersson. Unbounded stress in subregular phonology. In Garrett Nicolai and Ryan Cotterell, editors, *Proceedings of the 16th Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 135–143, Florence, Italy, 2019. Association for Computational Linguistics.

Bruce Hayes. *Metrical Stress Theory: Principles and Case Studies*. University of Chicago Press, 1995.

Jeffrey Heinz. On the role of locality in learning stress patterns. *Phonology*, 26(2):303–351, 2009.

Jeffrey Heinz. String extension learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 897–906, Uppsala, Sweden, 2010a. Association for Computational Linguistics.

Jeffrey Heinz. Learning long-distance phonotactics. *Linguistic Inquiry*, 41:623–661, 2010b.

Jeffrey Heinz. Computational Phonology – Part II: Grammars, Learning, Future. *Language and Linguistics Compass*, 5(4), 2011.

Jeffrey Heinz. Computational theories of learning and developmental psycholinguistics. In *The Oxford Handbook of Developmental Linguistics*, chapter 27. Oxford Academics, 2016.

Jeffrey Heinz. The computational nature of phonological generalizations. In Larry M. Hyman and Frans Plank, editors, *Phonological Typology*, volume 23 of *Phonology and Phonetics*, pages 126–195. De Gruyter Mouton, Berlin, Boston, 2018. doi: 10.1515/9783110451931-005.

Jeffrey Heinz and William Idsardi. Sentence and word complexity. *Science*, 333, 2011. doi: 10.1126/science.1210358.

Jeffrey Heinz and William Idsardi. What complexity differences reveal about domains in language. In *Topics in Cognitive Science*, volume 5, pages 111–131. Cognitive Science Society, 2013. doi: 10.1111/tops.12000.

Jeffrey Heinz and Regine Lai. Vowel harmony and subsequentiality. In András Kornai and Marco Kuhlmann, editors, *Proceedings of the 13th Meeting on the Mathematics of Language (MoL 13)*, pages 52–63, Sofia, Bulgaria, 2013.

Jeffrey Heinz, Chetan Rawal, and Herbert G. Tanner. Tier-based strictly local constraints for phonology. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, pages 58–64, 2011.

Jeffrey Heinz, Anna Kasprzik, and Timo Kötzing. Learning in the limit with lattice-structured hypothesis spaces. *Theoretical Computer Science*, 457:111–127, 2012. ISSN 0304-3975. doi: 10.1016/j.tcs.2012.07.017.

Thom Hess. *Dictionary of Puget Salish*. University of Washington Press, 1976.

Mans Hulden. *Finite-state machine construction methods and algorithms for phonology and morphology*. PhD thesis, University of Arizona, 2009.

Gerhard Jäager and James Rogers. Formal language theory: refining the chomsky hierarchy. *Philosophical Transactions of the Royal Society B*, 367, 2012. doi: 10.1098/rstb.2012.0077.

Roman Jakobson, Gunnar Fant, and Morris Halle. *Preliminaries to Speech Analysis*. MIT Press, 1952.

Adam Jardine. Computationally, tone is different. *Phonology*, 33:247–283, 2016a.

Adam Jardine. *Locality and Non-Linear Representations in Tonal Phonology*. PhD thesis, University of Delaware, 2016b.

Adam Jardine. The local nature of tone-association patterns. *Phonology*, 34:385–405, 2017. doi: 10.1017/S0952675717000185.

Adam Jardine. The expressivity of autosegmental grammars. *Journal of Logic, Language, and Information*, 28:9–54, 2019. doi: 10.1007/s10849-018-9270-x.

Adam Jardine and Jeffrey Heinz. Learning tier-based strictly 2-local languages. *Transactions of the Association for Computational Linguistics*, 4:87–98, 2016.

Adam Jardine and Chris Oakden. Computing process-specific constraints. *Linguistic Inquiry*, pages 1–9, 2023. doi: 10.1162/ling_a_00510.

Adam Jardine, Jane Chandlee, Rémi Eyraud, and Jeffrey Heinz. Very efficient learning of structured classes of subsequential functions from positive data. In *The 12th International Conference on Grammatical Inference*, volume 34, pages 94–108, 2014.

Douglas Johnson. *Formal aspects of phonological description*. De Gruyter, 1972.

Hyunjung Joo and Adam Jardine. Intonation as a quantifier-free logical interpretation of metrical and prosodic structure. In *Society for Computation in Linguistics*, volume 8, 2025. doi: https://doi.org/10.7275/scil.3124.

Aravind K. Joshi. Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural descriptions? In *Natural Language Parsing: Psychological, Computational, and Theoretical Perspectives*. Cambridge University Press, 1985.

Ronald M Kaplan and Martin Kay. Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–378, 1994.

Michael J. Kenstowicz and Charles W. Kisseberth. *Topics in Phonological Theory*. Academic Press, 1977.

Michael J. Kenstowicz and Charles W. Kisseberth. *Generative Phonology: Description and Theory*. New York : Academic Press, 1979.

Paul J. King. *A Logical Formalism for Head-Driven Phrase Structure Grammar*. PhD thesis, University of Manchester, 1989.

Nate Koser. *The Computational Nature of Stress Assignment*. PhD thesis, Rutgers University, 2022.

Nate Koser and Adam Jardine. The computational nature of stress assignment. In Hyunah Baek, Chikako Takahashi, and Alex Hong-Lun Yeung, editors, *Proceedings of the 2019 Meeting on Phonology*, Proceedings of the Annual Meetings on Phonology. Washington, DC: LSA, 2020.

Martin Krämer. *Vowel Harmony and Correspondence Theory*. De Gruyter Mouton, 2003. doi: 10.1515/9783110197310.

Nancy C Kula and Lee S Bickmore. Phrasal phonology in copperbelt bemba. *Phonology*, 32: 147–176, 2015.

Siki Kuroda. *Yawelmani Phonology*. MIT Press, 1967.

Regine Lai. Learnable vs. unlearnable harmony patterns. *Linguistic Inquiry*, 46(3):425–451, 2015. doi: 10.1162/LING_a_00188.

Dakotah Lambert and James Rogers. Tier-based strictly local stringsets: Perspectives from model and automata theory. In Allyson Ettinger, Gaja Jarosz, and Joe Pater, editors, *Proceedings of the Society for Computation in Linguistics 2020*, pages 159–166. Association for Computational Linguistics, 2020.

Dakotah Lambert, Jonathan Rawski, and Jeffrey Heinz. Typology emerges from simplicity in representations and learning. *Journal of Language Modeling*, 9, 2021.

Andrew Lamont. Sour grapes is phonotactically complex. In *LSA Annual Meeting 2019*, 2019.

Andrew Lamont. Phonotactics conspire to reduce computational complexity. unpublished manuscript, 2023.

Andrew Lamont, Charlie O'Hara, and Caitlin Smith. Weakly deterministic transformations are subregular. In Garrett Nicolai and Ryan Cotterell, editors, *Proceedings of the 16th Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 196–205, Florence, Italy, 2019. Association for Computational Linguistics.

Han Li. Learning tonotactic patterns over autosegmental. *Proceedings of the 2023 and 2024 Annual Meetings on Phonology*, 2025. doi: 10.7275/amphonology.3034.

Leonid Libkin. *Elements of Finite Model Theory*. Springer, 2004.

Huan Luo. Long-distance consonant harmony and subsequentiality. *Glossa*, 2:1–25, 2017.

David Marker. *Model Theory: An Introduction*. Springer, 2002.

Magdalena Markowska and Jeffrey Heinz. Empirical and theoretical arguments for using properties of letters for the learning of sequential functions. In *Proceedings of Machine Learning Research*, volume 217, page 270–274, 2023.

John McCarthy. *Derivations and Levels of Representation*, chapter 5. Cambridge University Press, 2007.

John J McCarthy. Process-specific constraints in optimality theory. *Linguistic inquiry*, 28:231–251, 1997.

Adam McCollum and James Essegbey. Unbounded harmony is not always myopic: Evidence from tutrugbu. In *Proceedings of the 35th West Coast Conference on Formal Linguistics*, pages 251–258, 2018.

Adam McCollum and Adam Jardine. Input- and output-oriented generalizations in iny atr harmony. unpublished manuscipt, 2022. URL https://lingbuzz.net/lingbuzz/006830.

Adam McCollum, Eric Baković, Anna Mari, and Eric Meinhardt. The expressivity of segmental phonology and the definition of weak determinism. technical report, 2018. URL https://ling.auf.net/lingbuzz/005565.

Adam McCollum, Eric Baković, Anna Mari, and Eric Meinhardt. Unbounded circumambient patterns in segmental phonology. *Phonology*, 37(2):215–255, 2020.

Kevin McMullin and Gunnar Hansson. Long-distance phonotactics as tier-based strictly 2-local languages. In *Proceedings of the 2014 Annual Meeting on Phonology*, 2016. doi: 10.3765/amp.v2i0.3750.

Kevin McMullin and Gunnar Ólafur Hansson. Inductive learning of locality relations in segmental phonology. *Laboratory Phonology*, 10(1), 2019. doi: 10.5334/labphon.150.

Eric Meinhardt, Anna Mai, Eric Baković, and Adam McCollum. On the proper treatment of weak determinism: Subsequentiality and simultaneous application in phonological maps. *UC San Diego Linguistic Papers*, 2021.

Eric Meinhardt, Anna Mai, Eric Baković, and Adam McCollum. Weak determinism and the computational consequences of interaction. *Natural Language and Linguistic Theory*, 42:1191 – 1232, 2024. doi: 10.1007/s11049-023-09578-1.

Stoyan Mihov and Klaus U. Schulz. *Finite-State Techniques: Automata, Transducers and Bimachines*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2019. doi: 10.1017/9781108756945.

Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational linguistics*, 23(2):269–311, 1997.

Yiannis Moschovakis. *Abstract Recursion and Intrinsic Complexity*, volume 48 of *Lecture Notes in Logic*. Cambridge University Press, 2019. doi: 10.1017/9781108234238.

Scott Nelson. A model theoretic perspective on phonological feature systems. In *Proceedings of the Society for Computation in Linguistics*, volume 5, 2022.

Scott Nelson and Eric Baković. Underspecification without underspecified representations. In *Society for Computation in Linguistics*, volume 7, pages 352—356, 2024. doi: 10.7275/scil.2227.

Chris Oakden. *Modeling phonological interactions using recursive schemes*. PhD thesis, Rutgers University, 2021.

David Odden. Principles of stress assignment: A crosslinguistic view. *Studies in the Linguistic Sciences*, 9(1):157—-176, 1979.

José Oncina, Pedro García, and Enrique Vidal. Learning subsequential transducers for pattern recognition interpretation tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15:448–458, 1993.

Henry A. Osborn. Warao i: Phonology and morphophonemics. *International Journal of American Linguistics*, 32(2), 1996.

Jaye Padgett. Partial class behavior and nasal place assimilation. In Keiichiro Suzuki and Dirk Elzinga, editors, *Proceedings of the 1995 Southwestern Workshop on Optimality Theory*. University of Arizona: Coyote Papers, 1995.

Joe Pater. *NC. In *Proceedings of the North East Linguistic Society*, volume 26, 1996.

Joe Pater. Austronesian nasal substitution and other nc effects. In René Kager, Harry van der Hulst, and Wim Zonneveld, editors, *The Prosody-Morphology Interface*, page 310–343. Cambridge University Press, 1999.

Joe Pater. Substance matters: a reply to jardine (2016). *Phonology*, 35(1):151–156, 2018. doi: 10.1017/S0952675717000409.

Amanda Payne. All dissimilation is computationally subsequential. *Language*, 93(4):353–371, 2017.

Sarah Payne. A generalized algorithm for learning positive and negative grammars with unconventional string models. In *Proceedings of the Society for Computation in Linguistics (SCiL)*, pages 75–85, 2024.

Long Peng. Nasal harmony in three south american languages. *International Journal of American Linguistics*, 66(1):76–97, 2000.

Glyne Piggott and Harry van der Hulst. Locality and the nature of nasal harmony. *Lingua*, 103: 85–112, 1997. doi: 10.1016/S0024-3841(97)00014-4.

William Poser. Phonological representation and action-at-a-distance. In Harry van der Hulst and Norval Smith, editors, *The Structure of Phonological Representations*. De Gruyter, 1982. doi: 10.1515/9783112423325-005.

Christopher Potts and Geoffrey K. Pullum. Model theory and the content of ot constraints. *Phonology*, 19(3):361–393, 2002. doi: 10.1017/S0952675703004408.

Geoffrey Keith Pullum and Barbara C. Scholz. On the distinction between model-theoretic and generative-enumerative syntactic frameworks. In Philippe de Groote, Glyn Morrill, and Christian Retoré, editors, *Logical Aspects of Computational Linguistics*, pages 17–43, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

Jonathan Rawski. *Structure and Learning in Natural Language*. PhD thesis, Stony Brook University, 2021.

James Rogers. A descriptive approach to language-theoretic complexity. *Computational Linguistics*, 27(2):304–308, 1998. doi: 10.1162/coli.2000.27.2.304.

James Rogers. Syntactic structures as multi-dimensional trees. *Research on Language and Computation*, 1:265–305, 2003. doi: 10.1023/A:1024695608419.

James Rogers and Dakotah Lambert. Some classes of sets of structures definable without quantifiers. In *Proceedings of the 16th Meeting on the Mathematics of Language*, pages 63–77. Association for Computational Linguistics, 2019. doi: 10.18653/v1/W19-5706.

James Rogers and Geoffrey Pullum. Aural pattern recognition experiments and the subregular hierarchy. *Journal of Logic, Language and Information*, 20:329–342, 2011. doi: 10.1007/s10849-011-9140-2.

James Rogers, Jeffrey Heinz, Jeremy Hurst, Dakotah Lambert, and Sean Wibel. Cognitive and sub-regular complexity. In *Lecture Notes in Computer Science*. Springer, 2013.

Sharon Rose and Racher Walker. A typology of consonant agreement as correspondence. *Language*, 80(3):475–531, 2004. doi: 10.1353/lan.2004.0144.

Edward Sapir and Morris Swadesh. *Yana Dictionary*. University of California Publications in Linguistics, 1960.

Paul Schachter and Victoria Fromkin. A phonology of akan: Akuapem, asante, fante. In *UCLA Working Papers in Phonetics*, volume 9. Phonetics Laboratory, University of California, 1968.

M.P. Schützenberger. A remark on finite transducers. *Information and Control*, 4(2):185–196, 1961. doi: 10.1016/S0019-9958(61)80006-5.

Dana Scott and Michael Rabin. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3:114 – 125, 1959. doi: 10.1147/rd.32.0114.

Stuart M. Shieber. Evidence against the context-freeness of natural language. In *The Formal Complexity of Natural Language*, pages 320 – 334. Springer, Dordrecht, 1985.

Kristina Strother-Garcia. Imdlawn Tashlhiyt Berber syllabification is quantifier-free. In *Proceedings of the Society for Computation in Linguistics*, page 145–153, 2018. doi: 10.7275/R5J67F4D.

Kristina Strother-Garcia. *Using model theory in phonology: a novel characterization of syllable structure and syllabification*. PhD thesis, University of Delaware, 2019.

Kristina Strother-Garcia and Jeffrey Heinz. Logical foundations of syllable representations. In *Proceedings of the Annual Meeting on Phonology*, 2017.

Kristina Strother-Garcia, Jeffrey Heinz, and Hyun Jin Hwangbo. Using model theory for grammatical inference: a case study from phonology. In *Proceedings of The 13th International Conference on Grammatical Inference*, pages 66–78. PMLR, 2016.

Bruce Tesar. *Output-Driven Phonology: Theory and Learning*. Cambridge University Press, 2013. doi: 10.1017/CBO9780511740039.

A. N. Tucker and J. T. O. Mpaayei. *A Maasai grammar, with vocabulary*. Longmans, Green, 1955.

Mai Vu, Ashkan Zehfroosh, Kristina Strother-Garcia, Michael Sebok, Jeffrey Heinz, and Herbert G. Tanner. Statistical relational learning with unconventional string models. *Frontier in Robotics and AI*, 2018. doi: 10.3389/frobt.2018.00076.

Rachel Walker. Prominence-driven stress. *Rutgers Optimality Archive*, 1996.

Janet Watson. The directionality of emphasis spread in arabic. *Linguistic Inquiry*, 30(2):289—-300, 1999. doi: 10.1162/002438999554066.

Colin Wilson. Analyzing unbounded spreading with constraints: Marks, targets, and derivations. unpublished manuscript, 2003.

William L. Wonderly. Zoque II: Phonemes and Morphophonemics. *International Journal of American Linguistics*, 17:105–123, 1951.