Local computation of cyclic phonology: A logical characterization of cyclic levels and cophonologies

Hossep Dolatian and Jeffrey Heinz

December 17, 2020

Abstract

Cyclicity is a common tool in phonology and morphology, but it is a difficult concept to formalize in computational systems. We develop a logical formalization for cyclic phonology and morphology, based on a case study in Armenian. Our formalization is based on a transformational and multistratal adaptation of One-Level Declarative Phonology. We call this system *Two-level Declarative Phonology*. Our logical formalism is flexible enough to operate over enriched hierarchical (tree-based) representations. The jump in representational power however is restricted by the generative capacity of the system. Most morphological and phonological processes are computationally local, i.e., quantifier-free in our logical system.

keywords: computational phonology, logic, declarative phonology, cyclicity, cyclic phonology, Armenian, cophonology, prosodic mapping, computational morphology

1 Introduction

Since Chomsky (1956), linguistics and computer science have often benefited from each other. Linguistics has provided case studies on the use of strings and trees as data structures, while computer science has provided formal grammars and learning frameworks (Miller and Chomsky 1963; Heinz et al. 2015; Pater 2019). Despite these connections, some linguistic concepts have resisted easy computational formalization. We focus on the problem of cyclicity in morphophonology.

Since Chomsky et al. (1956), generative phonology has utilized the Cycle. Informally, phonological rules can apply multiple times over the course of a derivation as morphemes are added. Since Kiparsky (1982), an additional concept has been Interactionism whereby morphological and

phonological processes are intermingled, oftentimes alongside prosodic representations (Booij and Lieber 1993). It is this intermingling that creates cyclic application of phonology.

Cyclicity is commonly employed in early and contemporary work in phonology and morphology (Scheer 2011). However, it is a difficult concept to computationally formalize and develop Sproat (1992:208). Intuitively, the problem is because of the following two challenges in finite-state treatments of phonology and morphology.

(1) *Challenges facing cyclic phonology*
   a. **Linearity**: Most computational systems work over linear representations, while morphophonological analyses often invoke nonlinear representations.
   b. **Boundedness**: A restrictive grammar cannot use the unbounded application of phonological rules. But the combination of cyclic rule application in phonology and of unbounded processes in morphology imply unbounded rule application.

In this paper, we develop a computational framework which avoids these two restrictions. Our formalization is based on formal logical transductions and model theory. We avoid Linearity because model theory is flexible enough to operate over a combination of strings, trees, and other data structures. This allows us to capture inherently derivational or transformational phenomena such as resyllabification, prosodic phonology (Nespor and Vogel 1986), cyclic levels (Bermúdez-Otero 2011), cophonologies (Inkelas 2014), and tree structure (Selkirk 1982).

By enriching our representations, we use our logical formalization to sidestep the problem of boundedness. We enrich our input representations with information on the derivational history of words (cf. Potts and Pullum 2002). This allows us to compute cyclic application and interactionism (Cole 1995), alongside counter-cyclic phenomena such as outwardly-sensitive allomorphy.

Our formalism is an adaptation of One-level Declarative Phonology (1-level DP) into a transformational framework. 1-level DP was a logical formalization of phonology which focused on using underspefication and feature-filling operations (Scobbie et al. 1996). It eschewed transformations in the traditional sense, and it was an inherently monotonic and monostratal system based on inviolable constraints. 1-level DP eventually gave way to Optimality Theory which combined violable constraints with transformational operations (Prince and Smolensky 2004).

In this paper, we enrich 1-level DP by incorporating logical transductions (Courcelle 1997). We call our formalism *Two-level Declarative Phonology* or 2-level DP. Logical transductions are the logical analogue of finite-state transducers. When defined over strings, logical transductions can exactly compute the classes of functions that are computed by finite-state systems. However, logic and model theory provide a flexible representational framework. We use logical transductions over non-string-based representations, similar to other model-theoretic approaches to language (Rogers 1998; Potts and Pullum 2002). Logical transductions are monotonic and declarative, but they are not monostratal: they transform an input representation into an output representation. 2-level DP is a recent branch of work in mathematical phonology (Jardine 2016; Strother-Garcia 2019; Dolatian 2020a; Dolatian et al. prep). Similar to OT, it is not a specific framework but a class of frameworks,

all tied together by the use of logical transductions.

This paper is based on Dolatian (2020a) who uses 2-level DP to develop a large-scale formalization of cyclic phonology in Armenian. Our goal is to showcase the utility of 2-level DP as an **explicit** framework to formalize cyclic phonology and morphology. We use 2-level DP to **define** a wide array of morphophonological phenomena. By being an explicit framework, 2-level DP enables us to examine the **generative capacity** of cyclic phonological processes over enriched hierarchical representations. The take-away is that, even with cyclicity and hierarchical representations, most of phonology and morphology are computationally local processes (cf. similar results over strings: Chandlee 2014; Heinz 2018; Chandlee and Heinz 2018; Jardine 2017a).

In §2 we go over the utility and problems behind finite-state systems. In §3, we explain the role of formal logic as a computational framework for studying phonology and morphology. We introduce 2-level Declarative Phonology. Before we showcase the utility of 2-level DP, we briefly go through a case study of cyclic phonology in Armenian (§4). The bulk of this paper is developed in §5. We define a fragment of a 2-level DP grammar for generating the cyclic phonology of Armenian, including aspects such as morphological structure, syllabification, prosodic parsing, strata, and cophonologies. In §6, we take a step back and evaluate our system. We show that it has wide empirical coverage (§6.1), can diagnose the locality of phonology (§6.2), and can integrate with the cyclicity of phonology (§6.3).

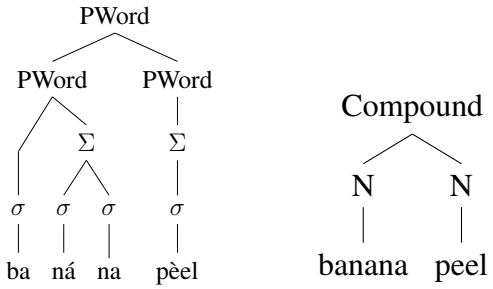
2 Finite-state systems and problems with boundaries

Cross-linguistically, languages can substantially differ in their morphophonological processes. But computationally, virtually all types of morphophonological processes are definable with finite-state calculus, specifically with 1-way finite-state acceptors (FSA) and transducers (FST) (Kaplan and Kay 1994; Beesley and Karttunen 2003; Roark and Sproat 2007).[1] This places a ceiling on the generative capacity of morphology and phonology.

FSTs are common formalism for computing phonological processes with ample implementational support (Roche and Schabes 1997; Hulden 2009; Gorman 2016). However, a problem with FSTs is that they are designed for linear representations. In contrast, phonological processes are often expressed over non-linear representations, specifically hierarchical representations in the form of morphological (2a) and prosodic trees (2b). When handled via a finite-state system, these structures are flattened down into a sequence of boundaries (2c). The finite-state machines then operate over a string of these segments and boundaries.

[1]The exception is total reduplication (Culy 1985). As a language, it needs multi-context free grammars; and as a function, it needs 2-way FSTs (Dolatian and Heinz 2020). 2-way FSTs are finite-state definable but more expressive than conventional 1-way FSTs (Engelfriet and Hoogeboom 2001).

(2) a. b.



c. $(_w(_w(_\sigma\text{ba})_\sigma(_\Sigma(_\sigma\text{ná})_\sigma(_\sigma\text{na})_\sigma)_\Sigma)_w(_w(_\Sigma(_\sigma\text{pèel})_\sigma)_\Sigma)_w)_w$

By using boundaries, FSTs can efficiently compute different morphological (Koskenniemi 1983; Beesley and Karttunen 2003) and prosodic processes (Hulden 2006; Idsardi 2009; Yu 2017, 2019). However, an open question is if these computational models act as faithful replicas of linguistic theory. For example, they may require placing a bound on the depth of the tree, require trees to all be right-branching, among other stipulations.[2] Because of these restrictions, the generative capacity of morphophonological processes when given hierarchical inputs is *different* than when given flattened inputs. For example, when tonal processes are defined in terms of strings, they can be computationally more complex, less local, and require more expressive grammatical formalisms than when these same processes are defined in more iconic representations like autosegmental grammars (Jardine 2016). Similarly, morphophonological processes often use local information over hierarchical representations, but this information can become non-local over strings.

Given this mismatch, this paper instead uses logical formulas to define and modify hierarchical representations for phonology and morphology. By doing so, we can faithfully formalize different linguistic phenomena, and then use our formalization to evaluate the generative capacity of these phenomena. The next section explains the use of formal logic in phonology

3 Logic and phonology

Although finite-state systems are a common computational notation for morphophonology, they are not the only one. In this section, we go over the use of logical systems for morphophonology. We go through the development of One-level Declarative Phonology, and then introduce a multi-stratal version of Declarative Phonology which we call two-level Declarative Phonology (§3.1). We illustrate 2-level DP in §3.2. Later in the paper, we use 2-level DP to formalize hierarchical structures in cyclic phonology, and we illustrate it with Armenian morphophonology

[2]Because of these issues, some use context-free grammars to capture hierarchical structure (Church 1983), with various restrictions so that they only generate regular languages.

3.1 Levels and Declarative Phonology

Declarative Phonology is a computational framework for theoretical phonology which developed in the early 1990's (Scobbie 1993). As a paradigm, it had two main characteristics. One was that it used logical statements in order to encode different morphophonological processes. The other is that it rejected the use of transformations or levels. Instead, DP formalized phonology as a monotonic and monostratal system based on the use of under-specification and well-formedness conditions (inviolable constraints).

Despite the reliance on under-specification on monotonicity, DP was a successful paradigm. DP used monotonic logic to formalize diverse types of morphophonological phenomena (Wheeler 1981; Scobbie 1991; Bird 1995; Coleman 1998; Chew 2003) . Multiple case studies were published in several conference proceedings (Ellison and Scobbie 1993; Bird et al. 1992) and journals (Bird and Klein 1994; Bird and Ellison 1994; Ogden 1999).

However, as a monostratal system, DP differed ontologically from mainstream phonological theory. As explained in more detail below, 1-level DP reduced transformations to under-specification and feature-filling, unlike mainstream phonology where transformations could *change* the underlying structure. In this paper, we incorporate transformations into a logical framework by using logical transductions couched within model theory (Büchi 1960; Courcelle 1994, 1997; Engelfriet and Hoogeboom 2001; Enderton 2001; Courcelle and Engelfriet 2012; Libkin 2013). By doing so, we formalize disparate types of morphophonological phenomena, including cases of cyclicity, phonological levels or strata, and resyllabification. Because of the combination of logic with transformations, we refer to our formalism as two-level Declarative Phonology or 2-level DP.

As a recent program, there is some work on using logical transductions for different morphophonological processes, including syllabification (Strother-Garcia 2018, 2019), tone (Chandlee and Jardine 2019a; Jardine 2016, 2017b; Koser et al. 2019), stress (Koser and Jardine 2019), and local and long-distance phonotactics (Chandlee and Jardine 2019b). Logical transductions have been used to evaluate different linguistic theories for their generative capacity or equivalence (Strother-Garcia 2019; Graf 2010; Danis and Jardine 2019; Jardine et al. 2020; Oakden 2020). There are learnability results (Strother-Garcia et al. 2017; Vu et al. 2018; Chandlee et al. 2019), and methods to convert logical transductions into neural network implementations (Rawski 2019) and to recursive schemata (Bhaskar et al. 2020).

This paper adds to this litany of developments in 2-level DP. We formalize a fragment of the cyclic phonology in Armenian. By doing so, we show how logical transductions (or 2-level DP) can formalize cyclicity, phonological domains, and morphological and prosodic structures.

3.2 Illustrating Declarative Phonology with 1 or 2 levels

Having explained the foundations for 1-level and 2-level DP, this section illustrates how 1-level and 2-level DP differ. We likewise set up the notation that we will use for the rest of the paper.

Consider a hypothetical language in (3a). It consists of only 3 surface allophones /p,b,a/. The voiced stop [b] is found word-initially while [p] is found elsewhere. A conventional phonological analysis is to posit that the two stops are allophones of a single phoneme. This analysis can be implemented in either a feature-changing or feature-filling format. In a feature-changing format (3b), the underlying phoneme is a voiceless /p/. Thus the lexical representation for [bapa] is /papa/. There is a rule which changes /p/ into a voiced segment [b] word-initially. In contrast in a feature-filling format (3c), the underlying phoneme is an archiphoneme /P/ that is underspecified for voicing. Thus the lexical representation for [paba] is /PaPa/. A feature-filling rule will add the feature [+voice] word-initially, and a separate rule will add the feature [-voice] elsewhere.

(3) a. paba aba pa ap

b. *Feature-changing*:
/papa/ /apa/ /pa/ /ap/
[+STOP, -VOICE] $\rightarrow$ [+STOP, +VOICE] / #_ $\sim$ /p/ $\rightarrow$ [b] / #_

c. *Feature-filling*:
/PaPa/ /aPa/ /Pa/ /aP/
[+STOP] $\rightarrow$ [+STOP, +VOICE] / #_ $\sim$ /P/ $\rightarrow$ [b] / #_
[+STOP] $\rightarrow$ [+STOP, -VOICE] / elsewhere $\sim$ /P/ $\rightarrow$ [p] / elsewhere

1-level and 2-level DP differ in this regard. We illustrate these approaches in (4), for generating the word [paba]. In terms of notation, the input consists of a domain D of elements, labels L over these elements to designate properties, and binary relations R between elements. The domain elements D are indexes {1,2,3,4} for each underlying segment. To visualize these representations, we use graphs where each node is a domain element identified with its index as a subscript.

(4) *Application of One-level and Two-level Declarative Phonology for hypothetical voicing*

One-level Declarative Phonology		Two-level Declarative Phonology	
Under-specified input	$P_1 \underset{\lhd}{\longrightarrow} a_2 \underset{\lhd}{\longrightarrow} P_3 \underset{\lhd}{\longrightarrow} a_3$	Input	$p_{0.1} \underset{\lhd}{\rightarrow} a_{0.2} \underset{\lhd}{\rightarrow} p_{0.3} \underset{\lhd}{\rightarrow} a_{0.4}$
Fully-specified output	$b_1 \underset{\lhd}{\longrightarrow} a_2 \underset{\lhd}{\longrightarrow} p_3 \underset{\lhd}{\longrightarrow} a_3$	Output	$b_{1.1} \underset{\lhd}{\rightarrow} a_{1.2} \underset{\lhd}{\rightarrow} p_{1.3} \underset{\lhd}{\rightarrow} a_{1.4}$

The labels L are the names of the IPA, such as $\mathsf{a}(x)$, general properties like $\mathsf{segment}(x)$, or names of binary features $+\mathsf{voice}(x)$.[3] We only show segment-based labels. The elements are connected via the binary relation of immediate precedence or successor $\mathsf{succ}(x, y)$ (shown as $\lhd$-labeled edges). The indexes for 2-level DP are explained below. Throughout this paper, we informally use the term 'node' to refer to both domain elements and to their graphical visualizations.

In (5), we show the logical formula used in a feature-filling 1-level DP analysis and a feature-changing 2-level DP analysis. Underlying labels and relations encode facts about the input: the segment *p* satisfies $+\mathsf{stop}(x)$. Complex conditions over the input are summarized in user-defined

[3]Technically, the relations R can be of any arity (Enderton 2001). What we call labels are simply unary relations.

predicates. For example, the predicate $\mathbf{initial}(x)$ (5a) finds the initial segment x which does not follow any other segment y. Because 1-level DP is a feature-filling formalism, the lexical representation for [paba] is underspecified as /PaPa/. A set of formulas (5b) fill in the appropriate voicing features based on context. In contrast, 2-level resembles feature-changing phonology. It posits an input level /papa/ and output level [bapa]. The output functions in (5c) add the feature `[+voice]` for initial $[-\mathsf{voice}]$ segments, while they faithfully output the feature `[-voice]` elsewhere. The notation for output functions in 2-level DP marks the variable with a superscript and is explained in more detail later.

(5) a. *Predicate for word-initial segments*
$\mathbf{initial}(x) \stackrel{\text{def}}{=} \mathsf{segment}(x) \wedge \neg\exists y[\mathsf{segment}(y) \wedge \mathsf{succ}(y, x)]$

b. *Feature-filling 1-level DP*
$+\mathsf{stop}(x) \wedge \mathbf{initial}(x) \rightarrow -\mathsf{voice}(x)$
$+\mathsf{stop}(x) \wedge \neg\mathbf{initial}(x) \rightarrow +\mathsf{voiced}(x)$

c. *Feature-changing 2-level DP*
$\texttt{+voiced}(x^1) \stackrel{\text{def}}{=} -\mathsf{voiced}(x) \wedge \mathbf{initial}(x)$
$\texttt{-voiced}(x^1) \stackrel{\text{def}}{=} -\mathsf{voiced}(x) \wedge \neg\mathbf{initial}(x)$

Formally, both 1-level DP and 2-level DP are monotonic and declarative. There is no specified order among the various logical statements, and they cannot contradict each other. However, 1-level DP is monostratal while 2-level is bistratal. In 1-level DP, the lexical representation persists; it cannot be changed and no destructive processes can be applied. Furthermore, the input and output are not different levels or strings. The output is simply the input with further explicit specifications. In contrast, 2-level DP posits an input level and output level which are in correspondence. The input undergoes transformations which can change, delete, or fill features and segments.

Unlike 1-level DP, 2-level DP allows the addition and deletion of symbols. To model insertion, 2-level DP uses a *copy set*, which indexes multiple corresponding copies of input domain elements. For example, consider a hypothetical case of final vowel epenthesis for consonant-final words: /ap/→[apa]. In 2-level DP, this process would require a logical transduction that uses a copy set of size 2: $C = \{1, 2\}$. As visualized in (6), Copy 1 is dedicated to outputting the input segments, while Copy 2 is dedicated to outputting the inserted vowel. Input domain elements D use indexes of the form $0.i$ where the right-index i is the index of the element in D. Output elements are distinguished based on the change in the left-index: $1.i$ or $2.i$.

(6) *Input and output for hypothetical final a-epenthesis*

Input	Output
$\mathrm{a}_{0.1} \underset{\triangleleft}{\rightarrow} \mathrm{p}_{0.2}$	*Copy 1* $\mathrm{a}_{1.1} \underset{\triangleleft}{\rightarrow} \mathrm{p}_{1.2}$ $\triangleleft \downarrow$ *Copy 2* $\mathrm{a}_{2.2}$

For generating output structures, 2-level DP treats the output nodes as correspondents of the input nodes. An input node can correspond to multiple output nodes. Correspondents are generated via output functions. These functions use the template $\texttt{label}(x^c)$ for labels and $\texttt{relation}(x^c, y^d)$ for binary relations, where c, d are the index of a copy. For final epenthesis, we use the predicate $\mathbf{final}(x)$ (7a) to find final segments. Over Copy 1, the output functions in (7b) faithfully generate all underlying labels L and relations R. We use a shorthand for listing out $\texttt{a}(x^1)$, $\texttt{succ}(x^1, y^1)$, etc. Over Copy 2 (7c), we output a vowel x^2 as an output correspondent of the final segment x *if* that segment x is a consonant. Visually, the epenthesized vowel $a_{2.2}$ is in correspondence with the final input segment $p_{0.2}$ because they share the same right-index *2*.

(7) a. *User-defined predicate to find final segments*
$\mathbf{final}(x) \stackrel{\text{def}}{=} \mathsf{segment}(x) \wedge \neg\exists y[\mathsf{segment}(y) \wedge \mathsf{succ}(x, y)]$

b. *Outputting the base faithfully in Copy 1*
- For every $\mathsf{label} \in L$:
$\texttt{label}(x^1) \stackrel{\text{def}}{=} \mathsf{label}(x)$
- For every $\mathsf{relation} \in R$:
$\texttt{relation}(x^1, y^1) \stackrel{\text{def}}{=} \mathsf{relation}(x, y)$

c. *Outputting the epenthetic vowel and connecting it with the base*
$\texttt{a}(x^2) \stackrel{\text{def}}{=} \mathsf{consonant}(x) \wedge \mathbf{final}(x)$
$\texttt{succ}(x^1, y^2) \stackrel{\text{def}}{=} \mathsf{segment}(x) \wedge \texttt{a}(y^2) \wedge (x = y)$

Having explained logical transductions, we can now apply them to a substantial fragment of a language. The next section first goes through a case study of cyclic phonology in Armenian.

4 Illustrating cyclicity with Armenian

Armenian is an agglutinative Indo-European language.We briefly introduce aspects of Armenian prosody, morphology, and phonology which we later formalize with 2-level DP. These aspects showcase the role of cyclicity and phonological rule domains.

Armenian is generally a CVCC language with final stress (8a) (Vaux 1998). If the final vowel is a schwa, stress is placed to the penultimate full vowel (8d). Phonologically, there is evidence that stress is assigned on a cyclic basis. When stressed high vowels lose stress, they are reduced in a process of destressed reduction (9). The high vowel is generally deleted (9a). But if deletion would create an unsyllabifiable consonant cluster, then the high vowel is reduced to a schwa (9b).

(8)
a.	kórd͡z	‘work’
b.	kord͡z-avór	‘worker’
c.	kord͡z-avor-nér	‘workers’
d.	kord͡z-avor-nér-ə	‘with workers’

(9)
a.	teɣín	‘yellow’
	teɣn-oráɡ	‘yellowish’
b.	hín	‘old’
	hən-utjún	‘oldness’

Destressed reduction targets high vowels which had stress in the base but not in the derivative. Thus, unstressed vowels in the base do not reduce in the derivative (10a). Furthermore, reduction can apply an unbounded number of times based on how much morphology we have (10b). It can apply in roots (10b), suffixes (10c), and in compounds (10d).

(10)

a.	məxitár	‘comforter’	c.	ázk	‘nation’
	məxitar-él	‘to comfort’		azk-ajín	‘national’
	*məxtar-él			azk-ajn-utjún	‘nationalist’
b.	d͡zín	‘birth’	d.	kír	‘handwriting’
	d͡zən-únt	‘birth’		kər-ít͡ʃ + dúp	‘pen + box’
	d͡zən-ənt-agán	‘generative’		kər-t͡ʃ-a-dup	‘pencil-box’

Morphologically, not every type of affix can trigger reduction. Derivational suffixes and compounding can trigger reduction (11i), while inflectional suffixes cannot (11ii).

(11)

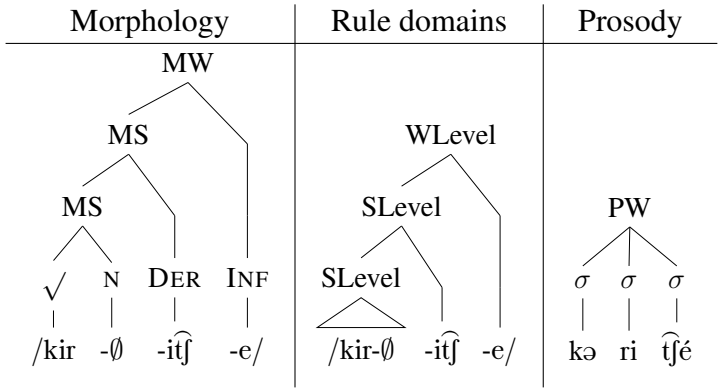
a.	i.	kír	‘writing’	b.	i.	amusín	‘husband’
		kər-ít͡ʃ	‘pen’			amusn-utjún	‘marriage’
	ii.	kər-it͡ʃ-é	‘pen-ABL’		ii.	amusin-óv	‘husband-INST’

Dolatian analyzes reduction with Lexical Phonology (Dolatian 2020a) and Stratal OT (Dolatian 2020b). Different types of morphology create different phonological domains. Free-standing roots and derivational morphology create morphological stems (MStems, MS).[4] These trigger the stem-level (SLevel) phonology of stress and reduction. In contrast, inflectional morphology creates morphological words (MWords, MW). They trigger the word-level phonology of stress without reduction. For example, we show the morphological tree for the inflected term *kər-it͡ʃ-e* (11a-ii), alongside a simplified prosodic tree.[5] We likewise provide a tree that showcases the different phonological rule domains of this word.

[4]To illustrate morphological functions, we assume that free-standing roots get their part of speech via a zero morpheme N (Marantz 2007). We don’t use lowercase letters *n* to avoid ambiguity. For simpler illustration, overt derivational and inflectional suffixes are labelled as DER and INF.

[5]Between syllables and prosodic words, Dolatian (2020a,b) uses an additional prosodic constituent called the prosodic stem (Downing 1999). We set this aside for illustration.

(12)



We illustrate a derivation for the inflected form *kər-itʃ-e*. Grey cells indicate inapplicable steps.

(13) *Derivation table for the cyclic derivation of an inflected derivative* kər-itʃ-e

Input				/kir -∅ -itʃ-e/
Cycle 1	MORPHO		Spell-out	kir -∅
	PROSODY		Syllabify	.kir.
	PHONO	*SLevel*	Stress	.kír.
			Reduction	
Cycle 2	MORPHO		Spell-out	.kír -/itʃ/
	PROSODY		Resyllabify	.kí.r-itʃ.
	PHONO	*SLevel*	Stress	.kǐ.r-ítʃ.
			Reduction	.kə.r-ítʃ.
Cycle 3	MORPHO		Spell-out	.kə.r-ítʃ -/e/
	PROSODY		Resyllabify	.kə.r-í.tʃ-e.
	PROSODY		Generate PWord	(.kə.r-í.tʃ-e)$_w$.
	PHONO	*WLevel*	Stress	.kə.r-i.tʃ-é.
Output				kər-itʃ-é

The root undergoes one cycle of the stem-level phonology to get stressed: *kír*. The derivational suffix triggers a new cycle of the stem-level phonology to get stressed and to reduce the root: *kər-ítʃ*. Between stress shift and reduction, we mark the destressed vowel with the diacritic *ǐ*. Finally, the inflectional suffix triggers a cycle of the word-level phonology to get stressed but without causing reduction: *kər-itʃ-é*. In this last cycle, we generate a prosodic word or PWord.

In sum, the Armenian data utilizes morphological and prosodic structure. These structures cyclically create phonological domains. The next section introduces an architecture for computing cyclic phonology and morphology using 2-level Declarative Phonology.

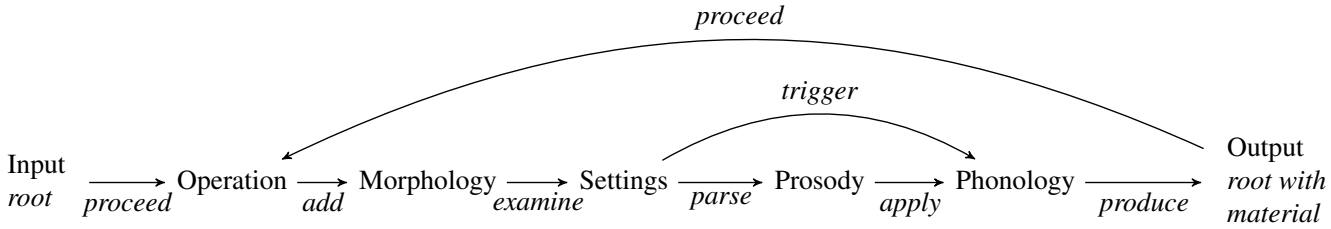
5 Application to cyclic phonology

As with any computational formalism, 2-level DP requires exact and well-defined representations and operations. In order to formalize cyclic phonology, we first set up the components of a cyclic architecture (§5.1). Using the Armenian word *kər-itʃ-e* as a running example, we then formalize the individual morphological, prosodic, and phonological operations across 3 cycles (§5.2-5.4).

5.1 Components in a cyclic architecture

In the previous section §4, we used an informal illustration in the form of a derivational table (13). This table explicitly encodes steps or components for morphology, prosody, and phonology. In order to formalize cyclic phonology, we develop an elaborated architecture in (14).

(14) *Elaborated architecture for cyclic phonology*



The figure in (14) shows the three expected modules of **Morphology**, **Prosody**, and **Phonology**. They are intermingled with two additional components: the **Operation** and the **Settings** (also called the **Encapsulation**). These two components are used to streamline the choice of morphological, prosodic, and phonological functions. Together, these 5 components comprise a single cycle. The output of this cycle is then fed back to itself to start a new cycle.

In a cycle, the first component is the **Operation**. It encodes the word's derivational order, i.e., a log of what morphological functions have applied or will be applied. This feeds the **Morphology** component which adds morphological structure. The morphology is then examined to find global properties of the input in the **Settings** or **Encapsulation** stage. These properties determine what prosodic parse to apply, and what phonological rule domains to trigger. These properties are encapsulated into a global constant called the SETTINGS. With the properties in place, we then apply the **Prosody** and **Phonology**.

Having explained the rough outline of our cyclic architecture, the next sections formalize the logical transductions which comprise these 5 components. Using *kər-itʃ-e* as our running example, we showcase these transductions over the course of three cycles.

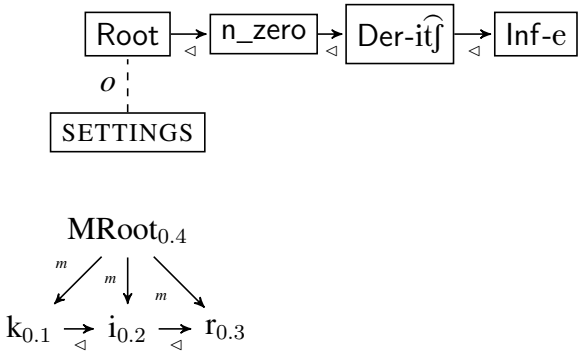
5.2 First cycle: Generating a stem

In the first cycle, the input is an unsyllabified and unstressed root $\sqrt{\text{kir}}$. The root lacks any part of speech. The output is a fully syllabified and stressed stem *kír*. To formalize this transformation, we first explain the input structure (§5.2.1). We process the derivational order (Operation list) in order to determine what morphological rules to apply (§5.2.2). We then apply the morphology (§5.2.3), encapsulation (§5.2.4), prosody of syllabification (§5.2.5), and phonological rule domains (§5.2.6).

5.2.1 Input: morphological and phonological representations

As said, the input to the first cycle is a simple root $\sqrt{\text{kir}}$. Phonologically, the segments of the root are in a linear order based on immediate successor. Morphologically, the segments are dominated by a root morpheme MRoot. The simple description is formalized in the figure in (15). The formalism combines a morphological tree with a linear phonological sequence of segments. Above the tree, we show two additional types of structures: the SETTINGS and **Operation List**. These structures encapsulate information about what morphological functions to apply.

(15) *Input to Cycle 1* – kir



For the phonology, the segments *k,i,r* have the labels $\mathsf{k}(x), \mathsf{i}(x), \mathsf{r}(x)$ respectively. Each segment likewise has the label $\mathsf{segment}(x)$. The segments are connected via the binary relation of immediate precedence or successor. The successor relation is specialized for only segments: $\mathsf{succ{:}seg}(x, y)$ (shown as $\lhd$-labeled edges). As for the morphology, the root has the labels $\mathsf{morpheme}(x)$ and $\mathsf{MRoot}(x)$. In the graphs, we only show the $\mathsf{MRoot}(x)$ label. The root is connected to its segments via the binary relation of morphological dominance $\mathsf{MDom}(x, y)$ (shown as *m*-labeled edges).[6]

Above the tree, the SETTINGS node is a constant (shown as a rectangle). The SETTINGS is connected to a sequence of nodes called the **Operation List**. The list encodes the derivational history of the word. Each of its nodes is called an **Operation** (shown in rectangles), and they have the label $\mathsf{oper}(x)$. Operation nodes are connected via the binary relation of successor, specialized for

[6] Dolatian (2020a) includes a morph level between segments and morphemes. We omit it for brevity.

operation nodes: succ:oper(x). The main label for each operation node is the name of the type of morphological function that it triggers. For our running example, the Operation List tells us that:

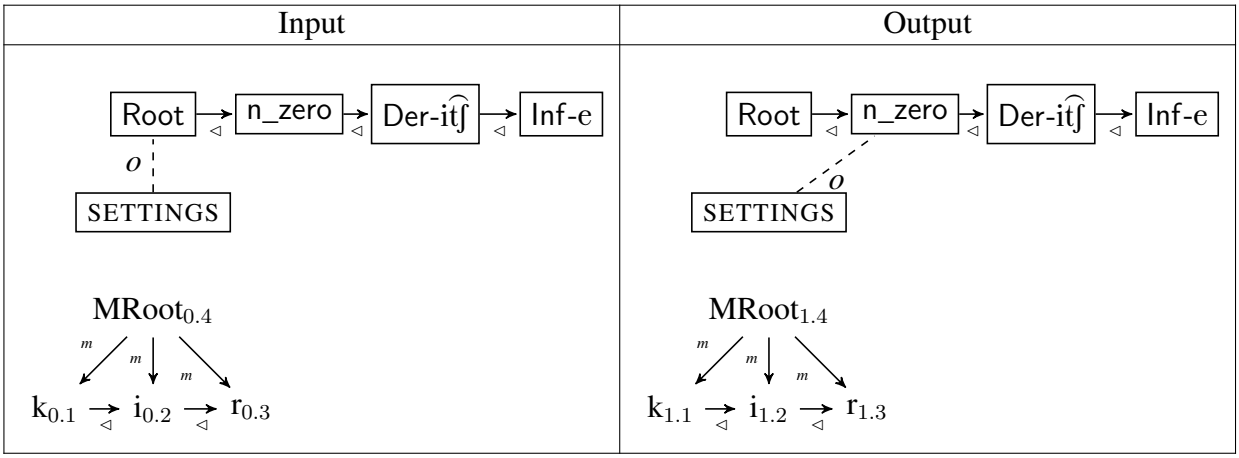
1. we start out with a root: oper:Root(x),
2. and we want to apply zero-nominalization: oper:n_zero(x),
3. followed by overt nominalization with the suffix -*itʃ*: oper:Der-itʃ(x),
4. and then inflection with the suffix -*e*: oper:Inf-e(x).

In the diagrams, we omit the prefix "oper:" from these operation labels. Finally, the SETTINGS is connected to the first operation via the binary relation of operate_at(SETTINGS, y) (shown as an o-labeled edge). In the diagram, we do not show indexes for the SETTINGS and operation nodes. They are all domain elements in the transduction.

5.2.2 Operation: Encoding derivational history

Given our input, the first cycle will start off by applying the Operation component. At this stage, we examine the input and its operation list. We determine what morphological function to apply in the subsequent Morphology component. Formally, the Operation component is a logical transduction that uses a copy set of size 1. We show the input and output below. In the input, SETTINGS constant is connected to the first operation node: oper:Root. In the output, the SETTINGS constant is reconnected with the subsequent operation node: oper:n_zero.

(16) *Operation: Proceeding on the operation list for the root* kir



In the Operation step, all labels and binary relations are faithfully outputted except for the operate_at(x, y) relation. Beyond this section, we do not show the formulas for faithfully outputting elements. They all follow the same format as in (17).

(17) *Faithfully output all labels and relations except for* operate_at(x, y)

- For every label $\in L$:

$\texttt{label}(x^1) \stackrel{\text{def}}{=} \mathsf{label}(x)$

- For every $\mathsf{relation} \in R - \{\mathsf{operate\_at}\}$:
  $\texttt{relation}(x^1, y^1) \stackrel{\text{def}}{=} \mathsf{relation}(x, y)$

To move along the operation list, we use the following predicates to find the current operation node $\mathsf{oper{:}Root}(x)$ which is connected to the SETTINGS (18a), and to find the subsequent operation node $\mathsf{oper{:}n\_zero}(x)$ (18b). The SETTINGS is then reconnected to the subsequent node (18c).

(18) *Predicates and output functions to move along the operation list*

a. $\mathbf{Oper{:}current}(x) \stackrel{\text{def}}{=} \mathsf{Oper}(x) \wedge \mathsf{operate\_at}(\text{SETTINGS}, x)$

b. $\mathbf{Oper{:}subsequent}(x) \stackrel{\text{def}}{=} \mathsf{Oper}(x) \wedge \exists y[\mathsf{Oper}(y) \wedge \mathbf{Oper{:}current}(y) \wedge \mathsf{succ{:}Oper}(y, x)]$

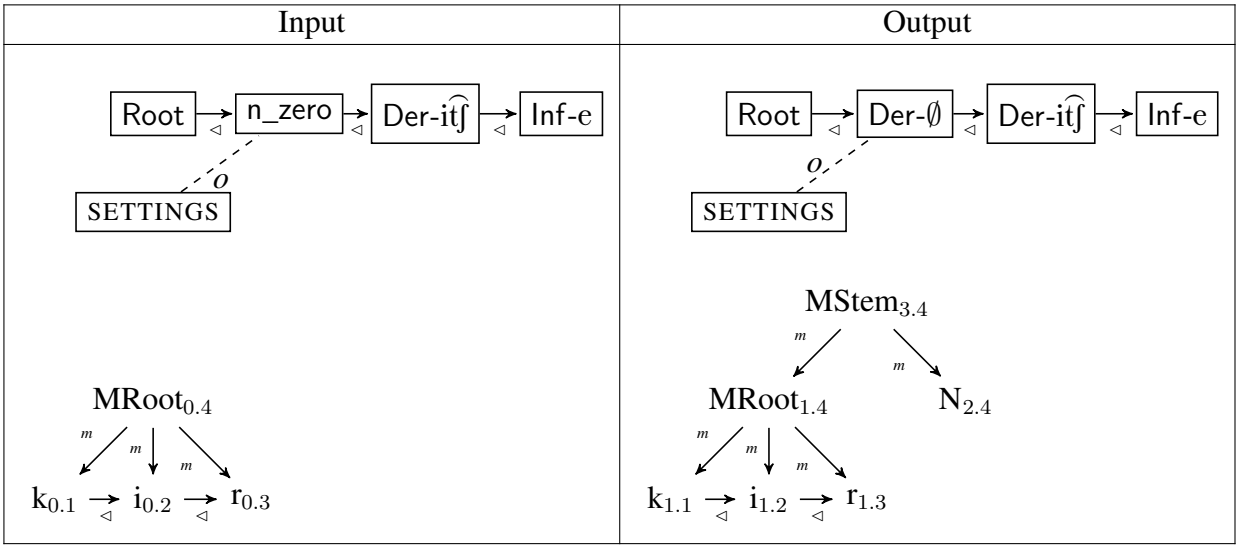
c. $\texttt{operate\_at}(\text{SETTINGS}^1, y^1) \stackrel{\text{def}}{=} \mathsf{Oper}(y) \wedge \mathbf{Oper{:}subsequent}(y)$

The SETTINGS constant now points to a new operation. With the operation list updated, the next stage is to apply the morphological function which corresponds to this new operation.

5.2.3 Morphology: Morphological functions

A language consists of a finite number of possible morphological processes which are applied in some order. We formalize processes as logical transductions that reference the operation list. We examine the label of the current operation node, and we apply the *type* of morphological process which this node is labeled for. For our running example, the morphology adds a zero affix -∅ to create an MStem. This requires logical transduction with a copy set of size 3.

(19) *Morphology: Adding a covert nominalizer in* kir-∅



We use the predicate in (20a) to check if the current operation node has the label oper:n_zero for zero-nominalization. We also use the predicate $\mathbf{MTopmost}(x)$ (20b) to find the morphologically-topmost node in the input, i.e., the morphological node which is at the top of the tree ($\text{MRoot}_{0.4}$).

(20) a. *Predicate to check if the current operation node triggers covert nominalization*
$\mathbf{CurrentOper}\text{:}\mathbf{n\_zero} \stackrel{\text{def}}{=} \exists y[\text{Oper}(y) \wedge \mathbf{Oper}\text{:}\mathbf{current}(y) \wedge \text{Oper:Der-}\emptyset]$

b. *Predicate to check for the morphologically topmost node*
$\mathbf{MTopmost}(x) \stackrel{\text{def}}{=} \text{MNode}(x) \wedge \neg\exists y[\text{MDom}(y, x)]$

Given these predicates, we then output the base *kir* in Copy 1 (not formalized here). We output the suffix morpheme $N_{2.4}$ in Copy 2 and the new $\textit{MStem}_{3.4}$ in Copy 3 as output correspondents of the morphologically-topmost node $\textit{MRoot}_{0.4}$ (21a). The morpheme has the labels $\text{N}(x)$ because it assigns part of speech, while the MStem has the labels $\text{MStem}(x)$. The MStem dominates its suffix and the base's MRoot via the output functions in (21b). We apply these functions only if we want to apply zero-nominalization, i.e., if the condition $\mathbf{CurrentOper}\text{:}\mathbf{n\_zero}$ is true

(21) *Predicates and output functions to generate a zero affix*

a. - $\texttt{N}(x^2) \stackrel{\text{def}}{=} \mathbf{MTopmost}(x) \wedge \mathbf{CurrentOper}\text{:}\mathbf{n\_zero}$
   - $\texttt{MStem}(x^3) \stackrel{\text{def}}{=} \mathbf{MTopmost}(x) \wedge \mathbf{CurrentOper}\text{:}\mathbf{n\_zero}$

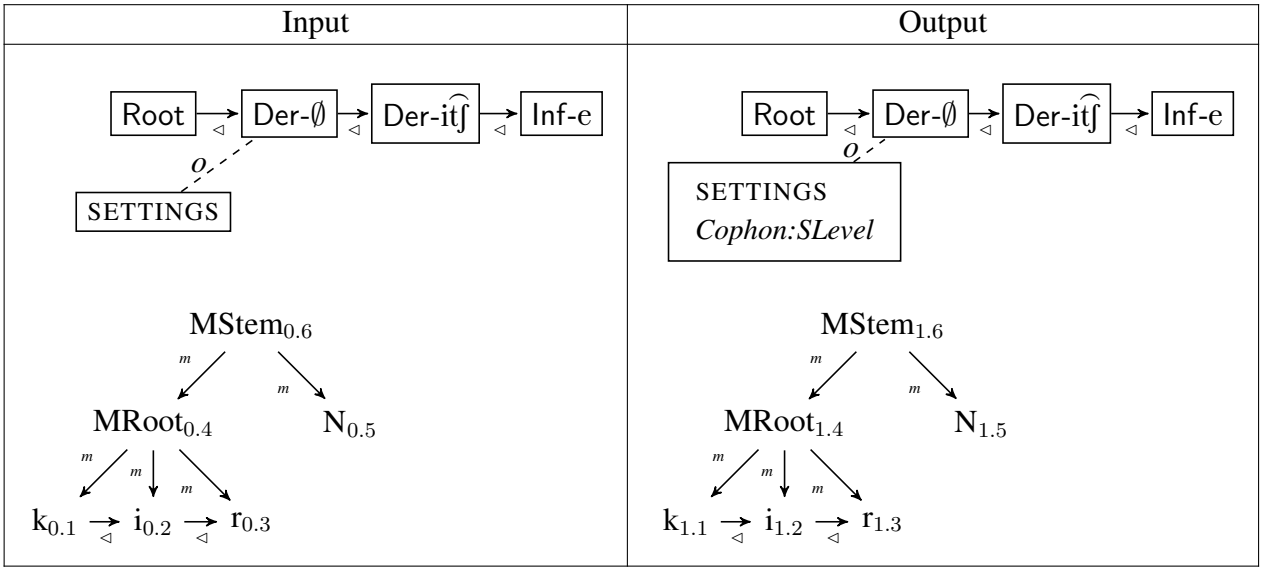
b. - $\texttt{MDom}(x^3, y^2) \stackrel{\text{def}}{=} \mathbf{MTopmost}(x) \wedge \mathbf{MTopmost}(y) \wedge \mathbf{CurrentOper}\text{:}\mathbf{n\_zero}$
   - $\texttt{MDom}(x^3, y^1) \stackrel{\text{def}}{=} \mathbf{MTopmost}(x) \wedge \mathbf{MTopmost}(y) \wedge \mathbf{CurrentOper}\text{:}\mathbf{n\_zero}$

5.2.4 Encapsulation: What to parse and what rules to apply

Following the Morphology, the next stage is Encapsulation. Encapsulation is a logical transduction with a copy set of size 1. In Encapsulation, we examine the morphological structure and determine what prosodic parses and what phonological rules to apply. This is done by examining the morphological and prosodic context of the morphologically-topmost node. Based on this context, we encapsulate properties onto the SETTINGS node. We show the input and output below.[7]

[7]The indexes are updated at the end of every transduction , e.g., the MStem now has the index 0.6 instead of 2.4.

(22) *Encapsulation: Setting the instructions for the prosody and phonology* kir



For the root *kir*, the morphology created an MStem and not an MWord. Thus we should not create a PWord in Cycle 1. We should apply the stem-level phonology, not the word-level phonology. In linguistic theory, the separation of stem-level and word-level phonology is formalized with separate cophonologies or levels (Inkelas 2014). We formalize cophonologies as *cophonology labels*: Cophon:SLevel(x) and Cophon:WLevel(x). We assume that all MStems and MWords automatically have these labels (23a). This label is percolated from the topmost node to the SETTINGS via the output functions in (23b). Our graphs show this label inside the SETTINGS box.

(23) a. *Encapsulation: Associating morphological constructions with cophonologies*
- MStem(x) $\rightarrow$ Cophon:SLevel(x)
- MWord(x) $\rightarrow$ Cophon:WLevel(x)

b. *Encapsulation: Assigning cophonology labels to the* SETTINGS
- `Cophon:SLevel`(SETTINGS^1) $\stackrel{\text{def}}{=} \exists x[\mathbf{MTopmost}(x) \wedge$ Cophon:SLevel(x)$]$
- `Cophon:WLevel`(SETTINGS^1) $\stackrel{\text{def}}{=} \exists x[\mathbf{MTopmost}(x) \wedge$ Cophon:WLevel(x)$]$

For *kir*, the SETTINGS inherits the stem-level feature from the topmost node *MStem*$_{0.6}$. With Encapsulation completed, we move on to the Prosody and Phonology.

5.2.5 Prosody: Syllabification

In the Prosody component, we apply syllabification, resyllabification, syllable linearization, and the generation of higher-level prosodic constituents like the prosodic word. These processes are each modeled by their own logical transduction which are composed into a single transduction.

For the stem *kir* in Cycle 1, the only appropriate prosodic process is syllabification.[8] So we only show syllabification. Formally, syllabification is a function with a copy set of size 2; the input and output are shown below, omitting any beyond the segments and syllables.

(24) *Prosody: Syllabifying the stem* kir

Input	Output
$\mathrm{k}_{0.1} \underset{\triangleleft}{\rightarrow} \mathrm{i}_{0.2} \underset{\triangleleft}{\rightarrow} \mathrm{r}_{0.3}$	$\mathrm{k}_{1.1} \underset{\triangleleft}{\rightarrow} \mathrm{i}_{1.2} \underset{\triangleleft}{\rightarrow} \mathrm{r}_{1.3}$ $\mathrm{k}_{1.1} \overset{p}{\leftarrow} \sigma_{2.2}$, $\mathrm{i}_{1.2} \overset{p}{\leftarrow} \sigma_{2.2}$, $\mathrm{r}_{1.3} \overset{p}{\leftarrow} \sigma_{2.2}$

Syllabification creates a single syllable over the stem *kir*. The syllable prosodically dominates the segments via different types of prosodic dominance relations which are specialized for the type of syllable position: $\mathsf{PDom{:}syll\_nuc}(x, y)$, $\mathsf{PDom{:}syll\_ons}(x, y)$, and $\mathsf{PDom{:}syll\_coda}(x, y)$.[9] In our graphs, we show these relations as simple *p*-labeled edges.

The first step in syllabification is to faithfully output all labels and relations in Copy 1 *except* for those that involve syllables and prosodic dominance (not shown). Following this, syllabification applies only to segments which are underlyingly unsyllabified. The predicate in (25a) checks if some segment is unsyllabified in the input. We likewise define a predicate (25b) that checks if a segment is unsyllabified over the output in Copy 1. The use of these two separate predicates is to simplify resyllabification in later cycles (§5.3.2). They both check if a segment x lacks a prosodic dominance relation to some syllable y.

(25) a. *Syllabification: Finding unsyllabified segments in the input*

$$\mathbf{unsyllabified}(x) \overset{\text{def}}{=} \neg\exists y[\mathsf{syll}(y) \wedge [\mathsf{PDom{:}syll\_ons}(y, x) \vee \mathsf{PDom{:}syll\_nuc}(y, x) \wedge \mathsf{PDom{:}syll\_coda}(y, x)]]$$

b. *Syllabification: Finding unsyllabified segments in the output, Copy 1*

$$\mathbf{unsyllabified}(x^1) \overset{\text{def}}{=} \neg\exists y[\mathtt{syll}(y^1) \wedge [\mathtt{PDom{:}syll\_ons}(y^1, x^1) \vee \mathtt{PDom{:}syll\_nuc}(y^1, x^1) \vee \mathtt{PDom{:}syll\_coda}(y^1, x^1)]]$$

With these predicates in hand, we will generate syllables and prosodic relations (26). For every vowel x^1 that is unsyllabified in Copy 1, we create a syllable in Copy 2 (26a). We then link any unsyllabified vowel y^1 to its newly created syllable x^2 as long as both x^2, y^1 are underlyingly the same vowel x in the input (26b).[10] As for onset formation (26c), we find any unsyllabified consonant y^1 in Copy 1 which precedes some vowel z^1, and then link the consonant y^1 to the

[8] As said in footnote (5), Dolatian (2020a) has MStems mapped to prosodic stems (PStem). For *kər-itʃ-e*, a PStem is created and modified in Cycles 1 and 2. We omit the PStem for brevity.

[9] We set aside complex codas. To model them, Dolatian (2020a) uses two types of coda-specific prosodic dominances: $\mathsf{PDom{:}syll\_coda1}(x, y)$ for the inner coda, and $\mathsf{PDom{:}syll\_coda2}(x, y)$ for the outer coda.

[10] Within a single transduction, we often use procedural terms like 'then'. This is only for ease of exposition. There is no fixed ordering between logical statements within a single transduction.

syllable x^2 of vowel z^1. Similarly for coda formation (26c), we find any consonant y^1 in Copy 1 which succeeds a vowel z^1 in Copy 1. We assign y^1 as the coda to the syllable x^2 of the vowel z^1, *if* the consonant y^1 isn't already the onset of some syllable u^2 in Copy 2.

(26) *Syllabification: Generating syllables and prosodic dominances*

a. $\texttt{syll}(x^2) \stackrel{\text{def}}{=} \texttt{vowel}(x^1) \wedge \mathbf{unsyllabified}(x^1)$

b. $\texttt{PDom:syll\_nuc}(x^2, y^1) \stackrel{\text{def}}{=} \texttt{syll}(x^2) \wedge \texttt{vowel}(y^1) \wedge \mathbf{unsyllabified}(y^1) \wedge (x = y)$

c. $\texttt{PDom:syll\_ons}(x^2, y^1) \stackrel{\text{def}}{=} \texttt{syll}(x^2) \wedge \texttt{consonant}(y^1) \wedge \mathbf{unsyllabified}(y^1) \wedge \exists z[\texttt{vowel}(z^1) \wedge \texttt{succ:seg}(y^1, z^1) \wedge \texttt{PDom:syll\_nuc}(x^2, z^1)]$

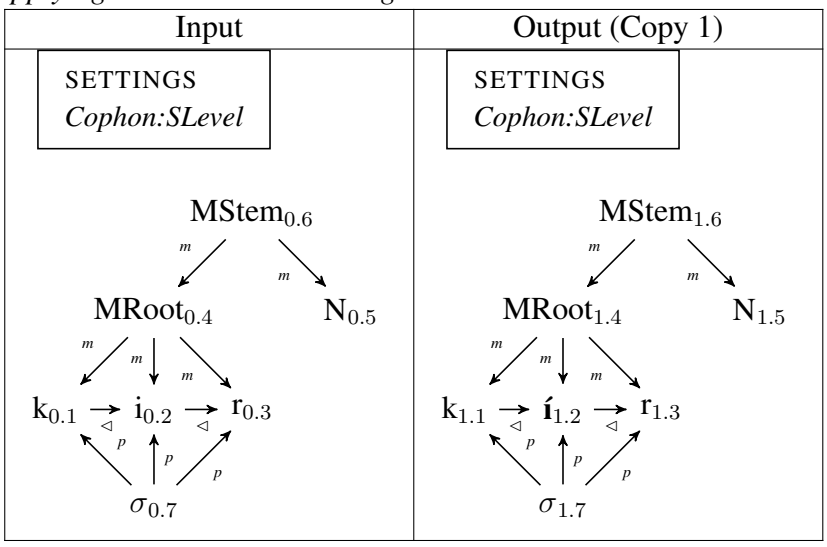
d. $\texttt{PDom:syll\_coda}(x^2, y^1) \stackrel{\text{def}}{=} \texttt{syll}(x^2) \wedge \texttt{consonant}(y^1) \wedge \mathbf{unsyllabified}(y^1) \wedge \exists z[\texttt{vowel}(z^1) \wedge \texttt{succ:seg}(z^1, y^1) \wedge] \wedge \texttt{PDom:syll\_nuc}(x^2, z^1)] \wedge \neg\exists u[\texttt{syll}(u^2) \wedge \texttt{PDom:syll\_ons}(u^2, y^1)]$

Following syllabification, multiple syllables are ordered or linearized. For monosyllabic *kir*, this step is vacuous. We postpone discussion till Cycle 2 (§5.3.2).

5.2.6 Phonology: Stem-level rule of stress

After the Prosody, the next stage is the Phonology. For the stem *kir*, the stem-level rule of stress applies. Reduction does not apply because there are no destressed vowels. We later formalize reduction in Cycle 2 (§5.3.3). We show the input and output below. The stressed vowel is in bold.

(27) *Stress: Applying stem-level stress assignment in* kir



Stress is a transduction with a copy set of size 1. Morphologically, stress is triggered if we are in the stem-level or word-level domains. The predicate (28) checks if we are in the right cophonology by examining the SETTINGS. We use this predicate as a condition on the application of stress.

(28) *Phonology: Finding the domain of stress*

- $\mathbf{StressDomain} \stackrel{\text{def}}{=} \mathsf{Cophon{:}SLevel}(\textsc{settings}) \vee \mathsf{Cophon{:}WLevel}(\textsc{settings})$

As said in §4, Armenian places stress on the rightmost full vowel. This is either the final syllable if it's a non-schwa; otherwise the penultimate syllable. We use the predicates in (29) to find these phonological contexts, i.e., full vowels, final syllables, and penultimate syllables which precede a schwa-headed syllable.[11]

(29) *Phonology: Predicates for finding the phonological context of stress*

- $\mathbf{vowel{:}full}(x) \stackrel{\text{def}}{=} \mathsf{vowel}(x) \wedge \neg\mathsf{schwa}(x)$
- $\mathbf{syll{:}final}(x) \stackrel{\text{def}}{=} \neg\exists y[\mathsf{succ{:}syll}(x, y)]$
- $\mathbf{syll{:}penult\text{-}pre\text{-}schwa}(x) \stackrel{\text{def}}{=} \mathsf{syll}(x) \wedge \exists y[\mathsf{succ{:}syll}(x, y) \wedge \mathbf{syll{:}final}(y) \wedge \mathbf{syll{:}schwa}(y)]$

With these predicates, we now apply stress. We faithfully output all labels and relations except for those which involve stress (not shown). Then we find the rightmost full vowel (30a), i.e., a vowel x which 1) is full, 2) part of the syllable y which is either 3) the final syllable or precedes a schwa-headed syllable. x is stressed by getting the label $\mathsf{stressed}(x)$ (30b). Crucially, the output function (30b) checks that we are the right morphologically-induced stress domain. This transduction is however incomplete. In Cycle 2 (§5.3.3), we add an output function to mark destressed vowels.

(30) *Phonology: Finding stressable vowels and applying stress*

a. $\mathbf{rightmost\_full\_vowel}(x) \stackrel{\text{def}}{=} \mathsf{vowel}(x) \wedge \mathbf{vowel{:}full}(x) \wedge$
$\exists y[\mathsf{syll}(x) \wedge \mathsf{PDom{:}syll\_nuc}(y, x) \wedge [\mathbf{syll{:}final}(x) \vee \mathbf{syll{:}penult\text{-}pre\text{-}schwa}(x)]]$

b. $\texttt{stressed}(x^1) \stackrel{\text{def}}{=} \mathbf{StressDomain} \wedge \mathsf{vowel}(x) \wedge \mathbf{rightmost\_full\_vowel}(x)$

This completes the generation of a free-standing noun *kír*. The output of this cycle is next fed back to the cyclic architecture in order to start a new cycle of morphology and phonology.

5.3 Second cycle: Generating a derivative

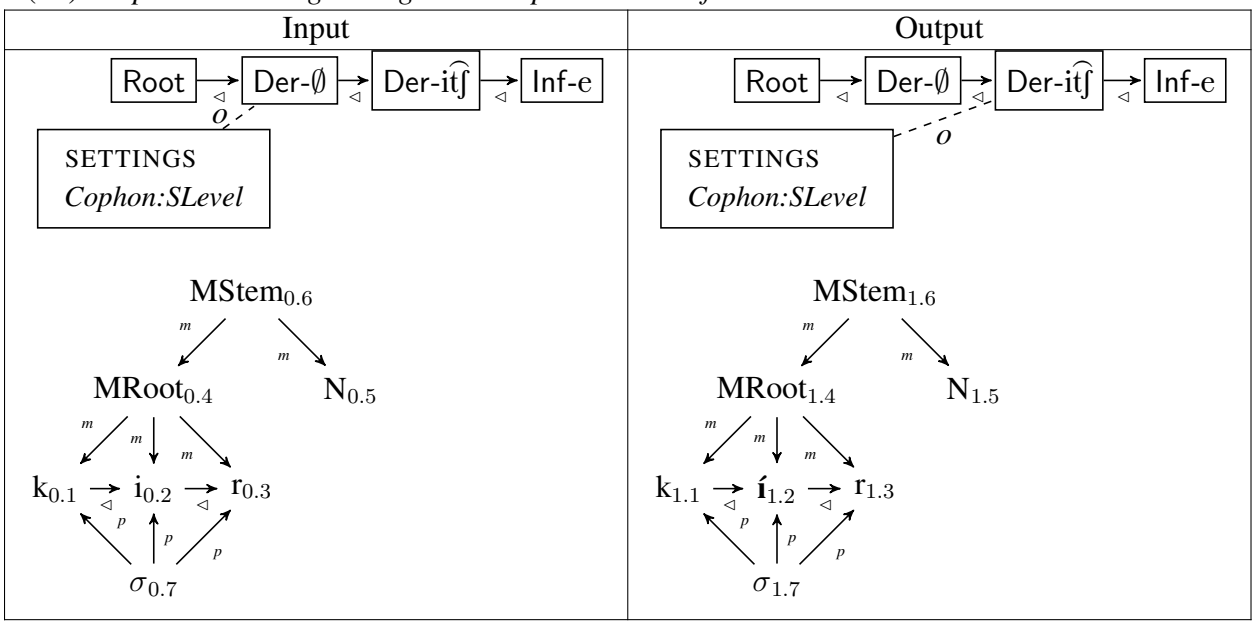
The first cycle showcased the basis of our logical formalization. In the second cycle, we show how to create overt morphological structure (§5.3.1), how to apply resyllabification (§5.3.2), and how to cyclically apply phonological rules (§5.3.3).

[11] Some of these predicates check for the linear order of syllables via the binary relation of syllable-based immediate successor: $\mathsf{succ{:}syll}(x, y)$, which we will discuss later in Cycle 2 (§5.3.2).

5.3.1 Operation and Morphology: Proceeding to add a derivational suffix

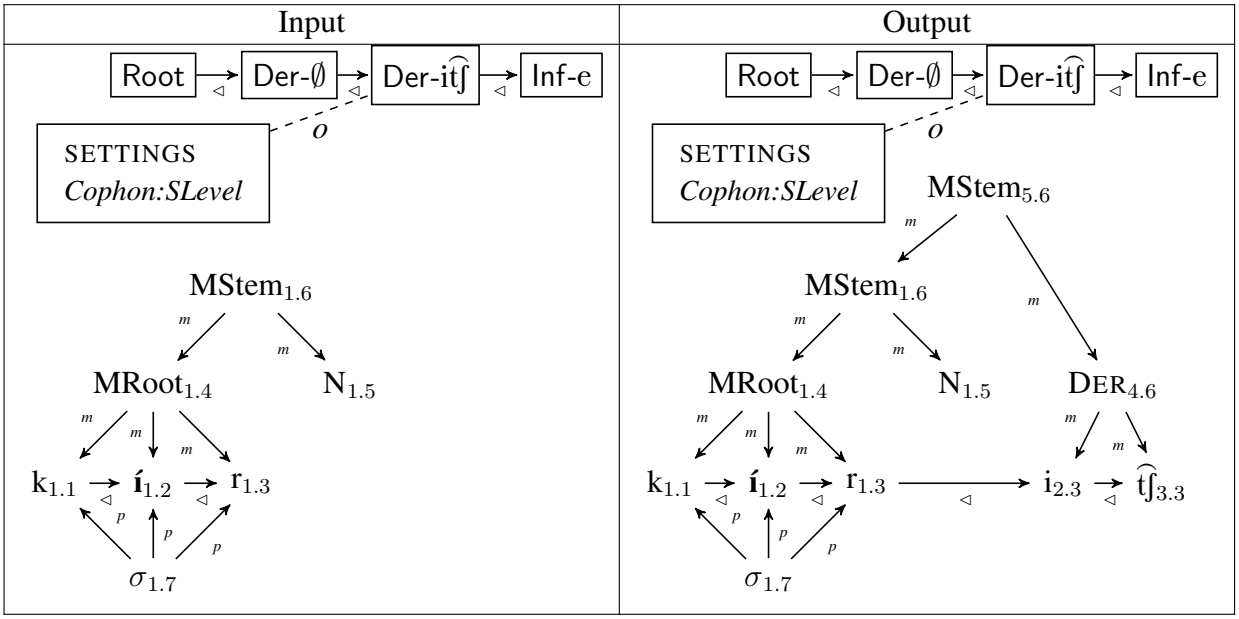
Given the stem as input to Cycle 2, we start in the Operation component. We progress on the operation list by shifting the SETTINGS to the subsequent operation oper:Der-$\widehat{\text{tʃ}}(x)$, i.e., adding a derivational suffix *-$\widehat{\text{itʃ}}$*. This processes uses the same output functions which we used for the Operation component in Cycle 1 (18, §5.2.2). We do not repeat these functions.

(31) *Operation: Progressing on the Operation List for a stem* kir



The output of the Operation component is fed to the Morphology. In the Morphology component in Cycle 1, we added a covert affix -∅. In Cycle 2, the Morphology component adds an overt derivational suffix *-$\widehat{\text{itʃ}}$* which creates a larger MStem. We show the input and output.

(32) *Input and output for adding an overt derivational suffix*



To generate this output, we need a logical transduction with a copy set of size 5, with the latter 4 copies reserved for every new segment. Note how this is different than the morphological transduction used for generating a zero affix in Cycle 1 (§5.2.3). The current transduction utilizes the predicate in (33a). The predicate examines the current current operation node on the Operation List in order to see if the node instructs us to add the derivational suffix *-it͡ʃ*. With this predicate, we output the base faithfully (not shown). As with zero-affixation, we output the suffix node and MStem node in Copies 4,5 as output correspondents of the morphologically-topmost node *MStem*$_{0.6}$ (33b). These two nodes are linearized together via morphological dominance (33c), and the new MStem$_{5.6}$ dominates the base's topmost MStem$_{1.6}$ (33d).

(33) *Morphology: Generating morphological nodes for an overt derivational suffix*

a. $\mathbf{CurrentOper}$:$\mathbf{Der}$-it͡ʃ$\stackrel{\text{def}}{=} \exists y[\mathsf{Oper}(y) \wedge \mathbf{Oper}$:$\mathbf{current}(y) \wedge \mathsf{Oper}$:Der-it͡ʃ$(y)]$

b. $\texttt{Der}(x^4) \stackrel{\text{def}}{=} \mathbf{MTopmost}(x) \wedge \mathbf{CurrentOper}$:$\mathbf{Der}$-it͡ʃ
$\texttt{MStem}(x^5) \stackrel{\text{def}}{=} \mathbf{MTopmost}(x) \wedge \mathbf{CurrentOper}$:$\mathbf{Der}$-it͡ʃ

c. $\texttt{MDom}(x^5, y^4) \stackrel{\text{def}}{=} \mathbf{MTopmost}(x) \wedge \mathbf{MTopmost}(y) \wedge \mathbf{CurrentOper}$:$\mathbf{Der}$-it͡ʃ

d. $\texttt{MDom}(x^5, y^1) \stackrel{\text{def}}{=} \mathbf{MTopmost}(x) \wedge \mathbf{MTopmost}(y) \wedge \mathbf{CurrentOper}$:$\mathbf{Der}$-it͡ʃ

The suffix segments *i,t͡ʃ* are defined as output correspondents of the input-final segment $r_{0.3}$ (34a). This segment is picked out by the predicate $\mathbf{final}(x)$ from §3.2 (7a). The suffix morpheme must dominate these suffix segments via morphological dominance (34b). The suffix segments are connected via successor (34c), and connected with the base via successor (34d).

(34) *Morphology: Generating segments for an overt derivational suffix*

a. $\texttt{i}(x^2) \stackrel{\text{def}}{=} \textbf{final}(x) \wedge \textbf{CurrentOper:Der-it}\widehat{\text{ʃ}}$
$\widehat{\text{tʃ}}(x^3) \stackrel{\text{def}}{=} \textbf{final}(x) \wedge \textbf{CurrentOper:Der-it}\widehat{\text{ʃ}}$

b. $\texttt{MDom}(x^4, y^2) \stackrel{\text{def}}{=} \textbf{MTopmost}(x) \wedge \textbf{final}(y) \wedge \textbf{CurrentOper:Der-it}\widehat{\text{ʃ}}$
$\texttt{MDom}(x^4, y^3) \stackrel{\text{def}}{=} \textbf{MTopmost}(x) \wedge \textbf{final}(y) \wedge \textbf{CurrentOper:Der-it}\widehat{\text{ʃ}}$

c. $\texttt{succ:seg}(x^2, y^3) \stackrel{\text{def}}{=} \textbf{final}(x) \wedge \textbf{final}(y) \wedge \textbf{CurrentOper:Der-it}\widehat{\text{ʃ}}$

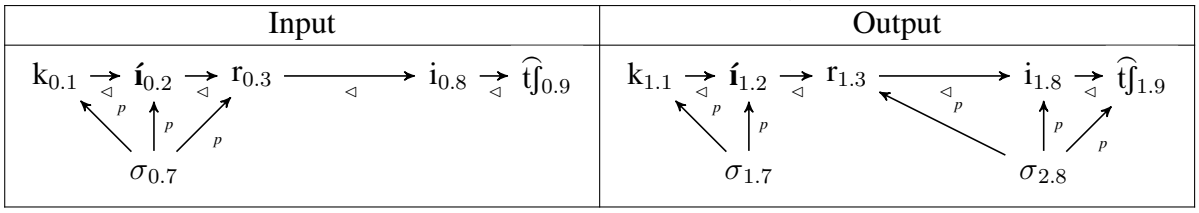
d. $\texttt{succ:seg}(x^1, y^2) \stackrel{\text{def}}{=} \textbf{final}(x) \wedge \textbf{final}(y) \wedge \textbf{CurrentOper:Der-it}\widehat{\text{ʃ}}$

In sum, overt affixation uses simple templates to generate and linearize segments and morphological nodes. The main difference among morphological functions is about 1) the different size of the copy set, and 2) the use of different conditions based on the Operation list.

5.3.2 Encapsulation and Prosody: Stem-levels and resyllabification

After the Morphology, the Encapsulation component applies again. The new topmost morphological node is the suffix MStem. Just like in Cycle 1, the SETTINGS gets the stem-level cophonology label via percolation from the topmost MStem node. The change is vacuous and we skip it. After Encapsulation, the Prosody will syllabify the suffix, and resyllabify the base's coda. We show this change below. For illustration, we only show the segments and syllables.

(35) *Prosody: Applying resyllabification to a derived stem* kír-itʃ



To handle resyllabification, we modify the output functions from Cycle 1 (§5.2.5). Copy 1 acts as a workspace to decide which underlying syllable relations will survive resyllabification and surface. The underlying syllable is faithfully outputted in Copy 1, alongside its nucleus and onset. As for codas, an underlying code faithfully surfaces *unless* the coda is before a vowel.[12]

(36) *Prosody: Faithfully outputting some but not all syllable structure*

a. $\texttt{syll}(x^1) \stackrel{\text{def}}{=} \textsf{syll}(x)$

b. $\texttt{PDom:syll\_nuc}(x^1, y^1) \stackrel{\text{def}}{=} \textsf{PDom:syll\_nuc}(x, y)$

c. $\texttt{PDom:syll\_ons}(x^1, y^1) \stackrel{\text{def}}{=} \textsf{PDom:syll\_ons}(x, y)$

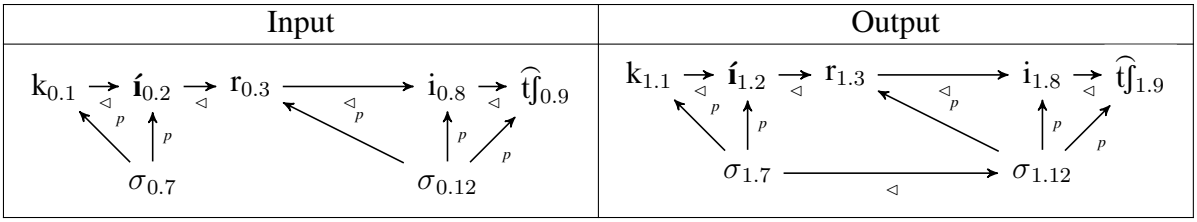
[12]Dolatian (2020a) provides additional conditions for resyllabification, such as when new consonants precede or follow the base's root, when new vowels precede the root, or in compound prosody. We set these aside.

d. $\texttt{PDom:syll\_coda}(x^1, y^1) \stackrel{\text{def}}{=} \textsf{PDom:syll\_coda}(x, y) \wedge \neg\exists z[\textsf{vowel}(z) \wedge \textsf{succ:seg}(y, z)]$

After these syllable structures are faithfully outputted in Copy 1, the suffix is newly syllabified in Copy 2. Syllabifying the suffix require the same output functions over Copy 2 as from Cycle 1 (§5.2.5). Those functions were illustrated for syllabifying a root, but they also work for syllabifying affixes. This is because those functions are defined over elements in Copy 1.

After (re)syllabification, the old and new syllables are ordered via a specialized type of immediate precedence or successor among syllables: $\textsf{succ:syll}(x, y)$. Syllable linearization is a transduction that uses a copy set of size 1. We show the input and output below.

(37) *Prosody: Ordering syllables in a derived stem* kír-itʃ͡



Syllables are ordered based on the general precedence relations of their nuclei. We first generalize the immediate successor relation of segments into general long-distance precedence (38b). We do this by determining the transitive closure of segment-based immediate successor (38a). Intuitively, the transitive closure utilizes a set X such that if some element x is in X, then so is every segment that comes after x. This creates a 'line' of immediate successor relations from x to the end of the input. With this new predicate for precedence, we generate a tier of vowels (38c). Syllables are ordered based on the location of their nuclei in the vowel tier (38d).[13]

(38) *Syllabification: Linearizing syllables with general precedence and tiers*

a. $\textbf{closed}\textbf{:succ:seg}(X) \stackrel{\text{def}}{=} \forall x, y[(x \in X \wedge \textsf{succ:seg}(x, y)) \rightarrow y \in X]$

b. $\textbf{gen\_prec:seg}(x, y) \stackrel{\text{def}}{=} \forall X[(x \in X \wedge \textbf{closed:succ:seg}(X)) \rightarrow y \in X] \wedge x \neq y$

c. $\textbf{tier\_local:vowel}(x, y) \stackrel{\text{def}}{=} \textsf{vowel}(x) \wedge \textsf{vowel}(y) \wedge \textbf{gen\_prec:seg}(x, y) \wedge \neg\exists w[\textsf{vowel}(w) \wedge \textbf{gen\_prec:seg}(x, w) \wedge \textbf{gen\_prec:seg}(w, y)]$

d. $\texttt{succ:syll}(x^1, y^1) \stackrel{\text{def}}{=} \textsf{syll}(x) \wedge \textsf{syll}(y) \wedge \exists u, v[\textsf{vowel}(u) \wedge \textsf{vowel}(v) \wedge \textsf{PDom:syll\_nuc}(x, u) \wedge \textsf{PDom:syll\_nuc}(y, v) \wedge \textbf{tier\_local:vowel}(u, v)]$

Having formalized resyllabification and syllable ordering, we now turn to cyclic phonological rules.

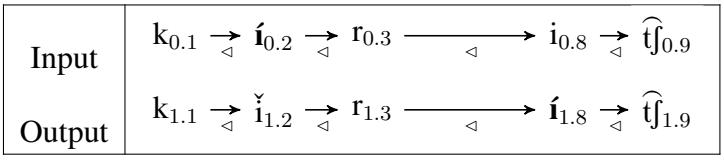
[13] These formulas uses Monadic Second Order (MSO) logic by using quantifiers over sets. Alternatively, we can turn general precedence into a primitive relation in our input (Heinz 2010); this would cause the above function to be First-Order (FO). Furthermore, because syllable sizes are bound (Strother-Garcia 2019), we don't actually need MSO/FO for syllable linearization but can use local quantifier-free logic. For illustration, we use MSO here.

5.3.3 Phonology: Cyclic reduction

Following the Prosody, we apply the Phonology. As with Cycle 1, we apply the stem-level cophonology. For Cycle 2, we apply both stem-level rules: stress and reduction. The Phonology component consists of two logical transductions which apply in a sequence: stress and reduction.

For stress, the same functions from Cycle 1 (§5.2.6) apply again. Stress is placed on the suffix vowel *-íʧ*, while the root vowel is marked as destressed *kĭr*. We show the input and output, but only the segments. We do not repeat the output functions for stress from Cycle 1 (§5.2.6).

(39) *Phonology: Applying stress shift in a derived stem* kĭr-íʧ

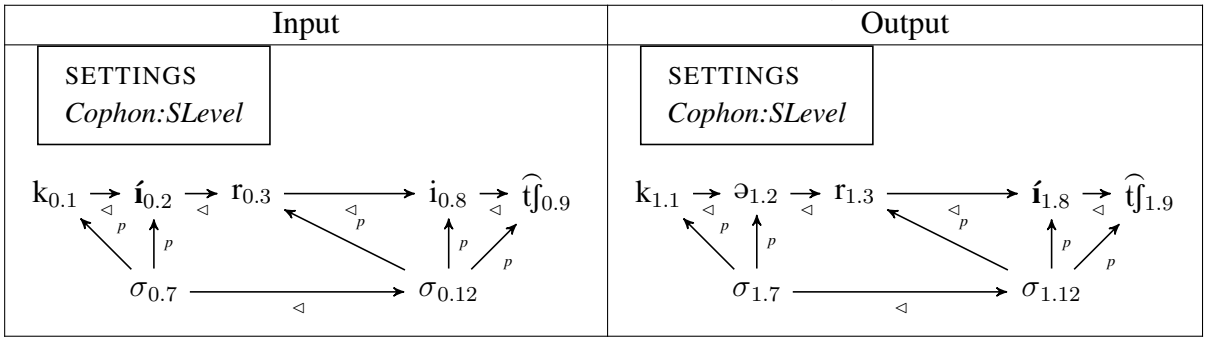


We introduce an output function for destressing (40). This function generates the label $\mathsf{destressed}(x)$ on a vowel which was stressed in the input but not the output.

(40) *Phonology: Applying destressing*
$\texttt{destressed}(x^1) \stackrel{\text{def}}{=} \mathsf{stressed}(x) \wedge \neg\texttt{stressed}(x^1)$

Following the application of stress shift, we apply vowel reduction. Reduction is modeled with a separate logical transduction that uses a copy set of size 1. We show the input and output below. We only show the segments, syllable nodes, and the SETTINGS constant.

(41) *Phonology: Applying vowel reduction to a derived stem* kər-íʧ



As explained in §4, destressed high vowels are either deleted or reduced to a schwa based on syllable structure. For illustration, we only show a partial formalization for reduction to schwa. See Dolatian (2020a) for a complete formalization. The transduction first checks that we are in the right morphological domain of reduction (42a), i.e., that we are in the stem-level domain based on the features of the SETTINGS constant. With this domain condition, we turn destressed high vowels into schwas (42b). Any underlying schwas are faithfully outputted.

(42) *Phonology: Finding the domain of reduction and applying reduction*

a. $\textbf{ReductionDomain} \stackrel{\text{def}}{=} \text{Cophon:SLevel}(\text{SETTINGS})$

b. $\texttt{schwa}(x) \stackrel{\text{def}}{=} \textbf{ReductionDomain} \wedge [\text{schwa}(x) \vee [\text{destressed}(x) \wedge \text{high}(x)]]$

This completes the second cycle. Our logical formalism exactly captures the fact that reduction is a cyclic process that is derived via multiple cycles of stress shift and reduction. In the next cycle, we show how our formalism separates the stem-level domain from the word-level domain.

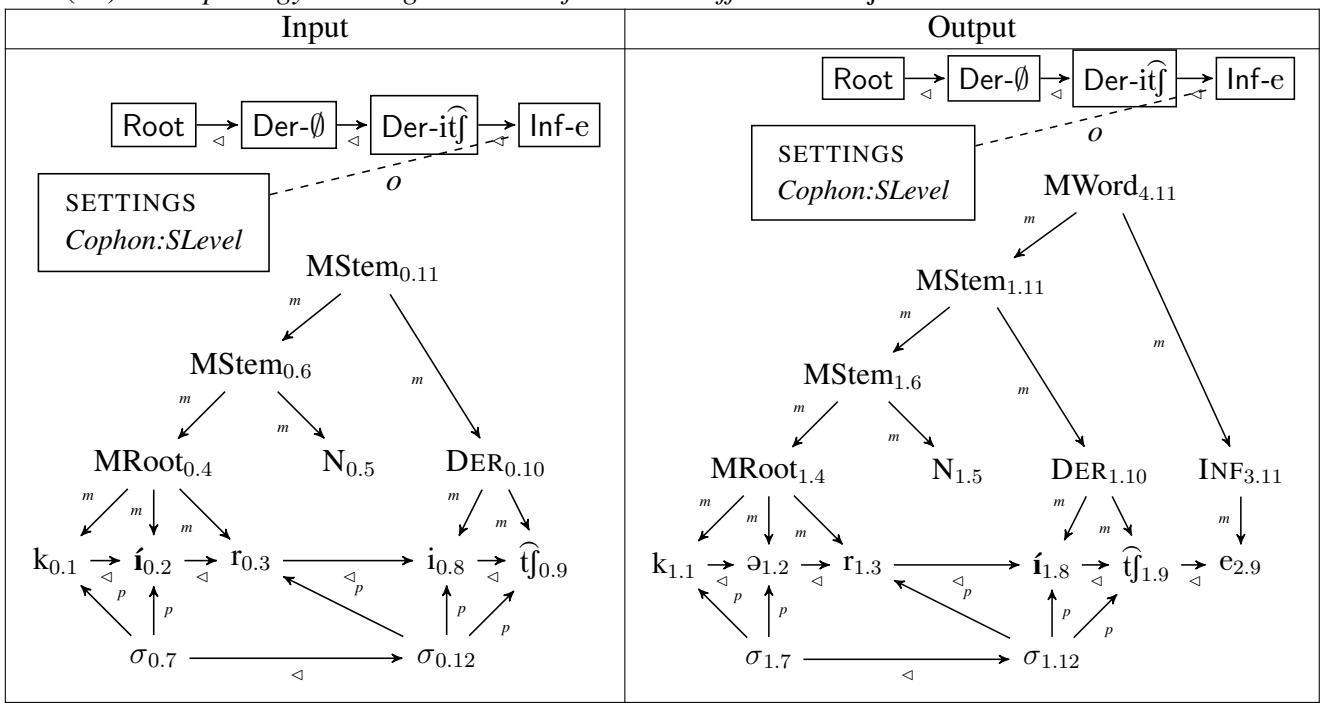
5.4 Third cycle: Word-level phonology

The output of the second cycle *kər-ít͡ʃ* is fed to a new third cycle to form an inflected word *kər-it͡ʃ-é*. The main aspects of this cycle are creating a different overt affix (§5.4.1), parsing prosodic words (§5.4.3), and applying a separate morphologically-induced cophonology (§5.4.4).

5.4.1 Operation to Morphology

In the third cycle, the Operation and Morphology components do the tasks of generating an overt inflectional affix. We first advance on the operation list so that the SETTINGS points to the final operation node: oper:Inf-e. This transduction uses the same functions from Cycle 1 (§5.2.2). We don't show the input-output of the Operation stage. We then move on to the Morphology where we add the inflectional suffix *-e*. The input and output are shown below.

(43) *Morphology: Adding an overt inflectional suffix in* kər-ít͡ʃ-e



As in previous Morphology stages, we utilize a predicate which checks if the current operation node instructs us to add an inflectional suffix -*e* (44a). With this condition, we apply a set of output functions which are specialized to generate the inflectional suffix -*e*. The transduction uses a copy set of size 4, with the latter 3 dedicated to the suffix. We output the base (not shown), the suffix segment (44b), the suffix morphology (44c), and then connect them all together via successor (44d) and morphological dominance (44e). As before, we check for the right operation condition.

(44) *Morphology: Generating an inflectional suffix* -e

a. $\mathbf{CurrentOper}$:$\mathbf{Inf}$-e$\stackrel{\text{def}}{=} \exists y[\mathsf{Oper}(y) \wedge \mathbf{Oper}$:$\mathbf{current}(y) \wedge \mathsf{Oper}$:Der-e$(y)]$

b. $\mathtt{e}(x^2) \stackrel{\text{def}}{=} \mathbf{final}(x) \wedge \mathbf{CurrentOper}$:$\mathbf{Inf}$-e

c. $\mathtt{Inf}(x^3) \stackrel{\text{def}}{=} \mathbf{MTopmost}(x) \wedge \mathbf{CurrentOper}$:$\mathbf{Inf}$-e
$\mathtt{MWord}(x^4) \stackrel{\text{def}}{=} \mathbf{MTopmost}(x) \wedge \mathbf{CurrentOper}$:$\mathbf{Inf}$-e

d. $\mathtt{succ:seg}(x^1, y^2) \stackrel{\text{def}}{=} \mathbf{final}(x) \wedge \mathbf{final}(y) \wedge \mathbf{CurrentOper}$:$\mathbf{Inf}$-e

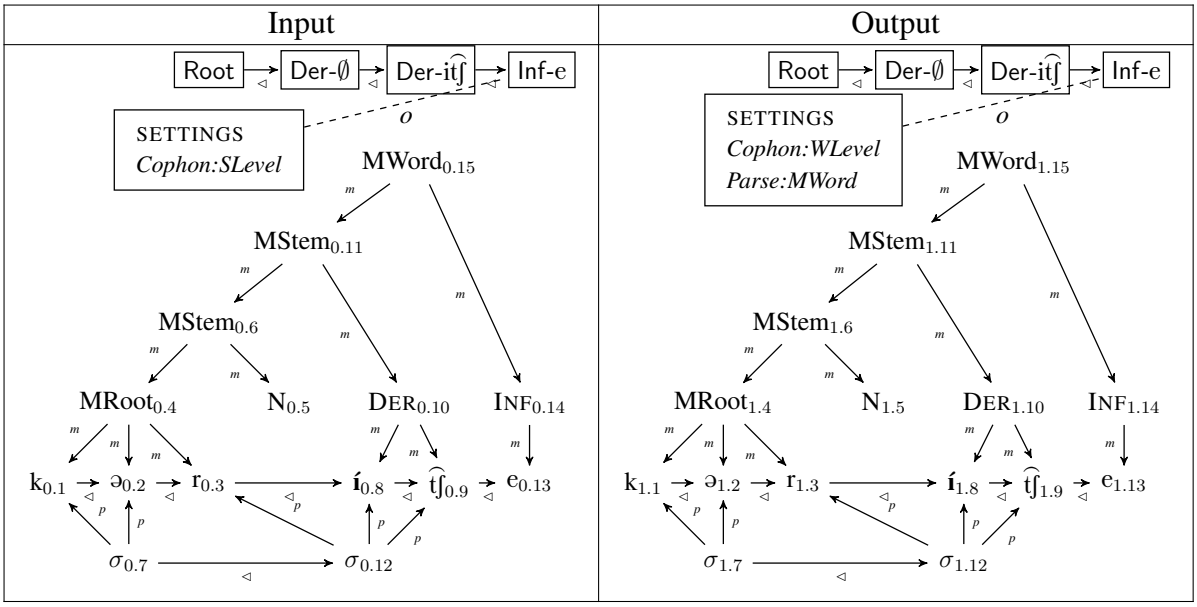
e. $\mathtt{MDom}(x^3, y^2) \stackrel{\text{def}}{=} \mathbf{MTopmost}(x) \wedge \mathbf{final}(y) \wedge \mathbf{CurrentOper}$:$\mathbf{Inf}$-e
$\mathtt{MDom}(x^4, y^3) \stackrel{\text{def}}{=} \mathbf{MTopmost}(x) \wedge \mathbf{MTopmost}(y) \wedge \mathbf{CurrentOper}$:$\mathbf{Inf}$-e
$\mathtt{MDom}(x^4, y^1) \stackrel{\text{def}}{=} \mathbf{MTopmost}(x) \wedge \mathbf{MTopmost}(y) \wedge \mathbf{CurrentOper}$:$\mathbf{Inf}$-e

For the morphology, Cycles 2 and 3 use two transductions with different copy sets (4,5) to generate the right suffixes. They both used different operation-based conditions on their application. See Dolatian (2020a) on how to make a unified grammar for morphological functions by using these operation-based conditions.

5.4.2 Encapsulation: Parsing instructions

The output of the Morphology is a morphological word. Unlike stems, words trigger the word-level cophonology and they are parsed into prosodic words. In order to apply these processes, in the Encapsulation stage, the SETTINGS constant is given the word-level cophonology label. It is likewise given a parsing instruction to trigger PWord formation. The output is shown below.

(45) *Encapsulation: Percolating word-level cophonology and prosodic parse for* kər-ít͡ʃ-e



The label for the word-level cophonology is percolated via the output function in (46a). As for the parse instructions (46b), this label is generated on the SETTINGS if the morphologically topmost node is an MWord which is not already parsed into a PWord. To encode this property, we use the binary relation of Match:word(x, y) which associates an MWord with a matching PWord. This relation is explained further in the Prosody stage (§5.4.3).

(46) *Encapsulation: Generating the word-level and PWord-parsing labels on the* SETTINGS

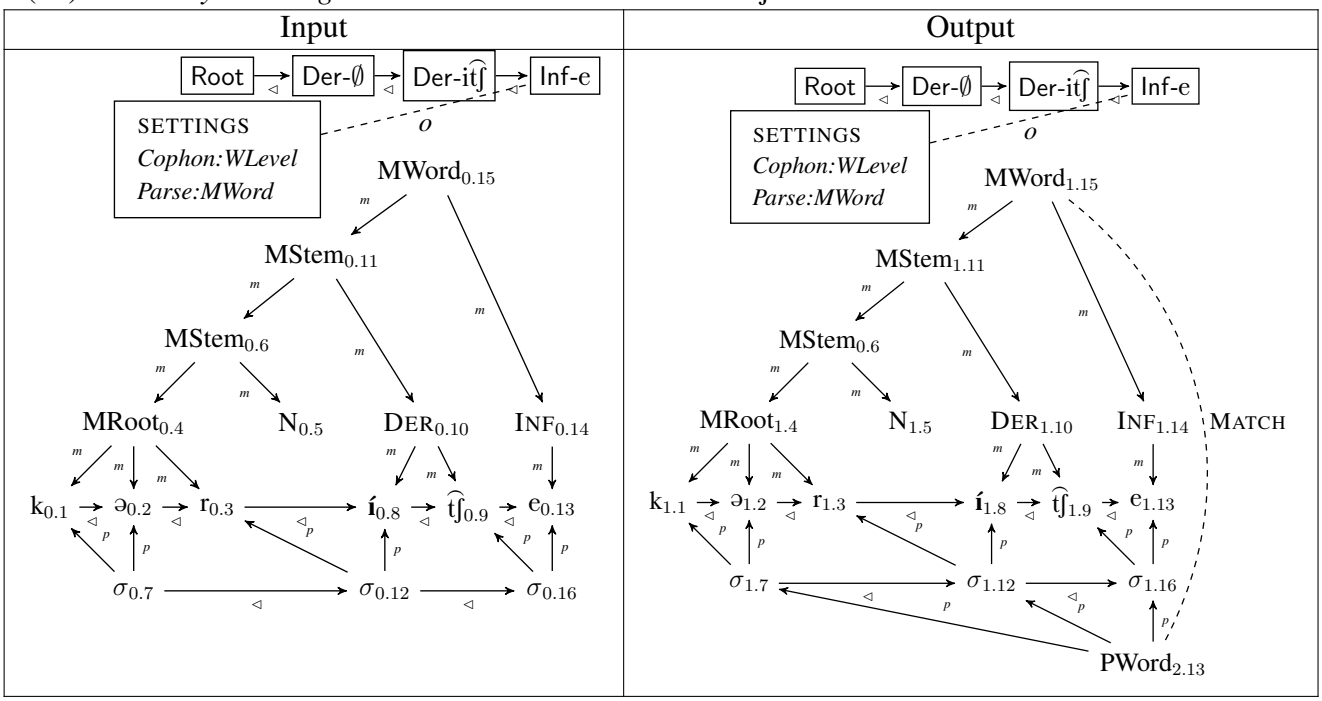
a. `Cophon:WLevel`(SETTINGS^1) $\stackrel{\text{def}}{=} \exists x[\mathbf{MTopmost}(x) \wedge \text{Cophon:WLevel}(x)]$

b. `Parse:MWord`(SETTINGS^1) $\stackrel{\text{def}}{=} \exists x[\mathbf{MTopmost}(x) \wedge \text{MWord}(x) \wedge \neg\exists w[\text{PWord}(w) \wedge \text{Match:word}(x, w)]]$

5.4.3 Prosody: Generating prosodic words

After Encapsulation, the Prosody applies. The Prosody involves resyllabification and generating higher-level prosodic structure. The suffix is first resyllabified with the base, using the same functions from Cycle 2 (§5.3.2): *.kə.rít͡ʃ.⟨e⟩→.kə.rí.t͡ʃe.* We do not show the input and output for resyllabification. We then examine the SETTINGS constant to check if we should generate any prosodic constituents. The SETTINGS has the label Parse:MWord and thus we are instructed to generate a PWord. We show the input and output below.

(47) *Prosody: Parsing an MWord into a PWord in* kər-ít͡ʃ-e



When parsing a MWord into an PWord, we use the SETTINGS to check if we have the right instruction to parse a PWord: Parse:MWord(SETTINGS). With this condition, the input is faithfully outputted in Copy 1 (not shown). The PWord is generated as an output correspondent of the input's topmost MWord (48a). Although redundant, we check that the MWord is unparsed in the input. The MWord and PWord are matched together (48b) via a special binary relation that's specialized for prosodic correspondences (Selkirk 2011). Finally, the PWord dominates the syllables (48c) via a type of prosodic dominance that's specialized for PWords and syllables.

(48) *Prosody: Generating a PWord*

a. $\texttt{PWord}(x^2) \stackrel{\text{def}}{=} \textsf{Parse:MWord}(\textsc{settings}) \wedge \textsf{MWord}(x) \wedge \neg\exists y[\textsf{PWord}(y) \wedge \textsf{Match:word}(x, y)]$

b. $\texttt{Match:word}(x^1, y^2) \stackrel{\text{def}}{=} \textsf{Parse:MWord}(\textsc{settings}) \wedge \texttt{MWord}(x^1) \wedge \texttt{PWord}(y^2) \wedge x = y \wedge \neg\exists z[\textsf{PWord}(z) \wedge \textsf{Match:word}(x, z)]$

c. $\texttt{PDom:PWord\_syll}(x^2, y^1) \stackrel{\text{def}}{=} \textsf{Parse:MWord}(\textsc{settings}) \wedge \texttt{PWord}(x^2) \wedge \texttt{syll}(y^1)$

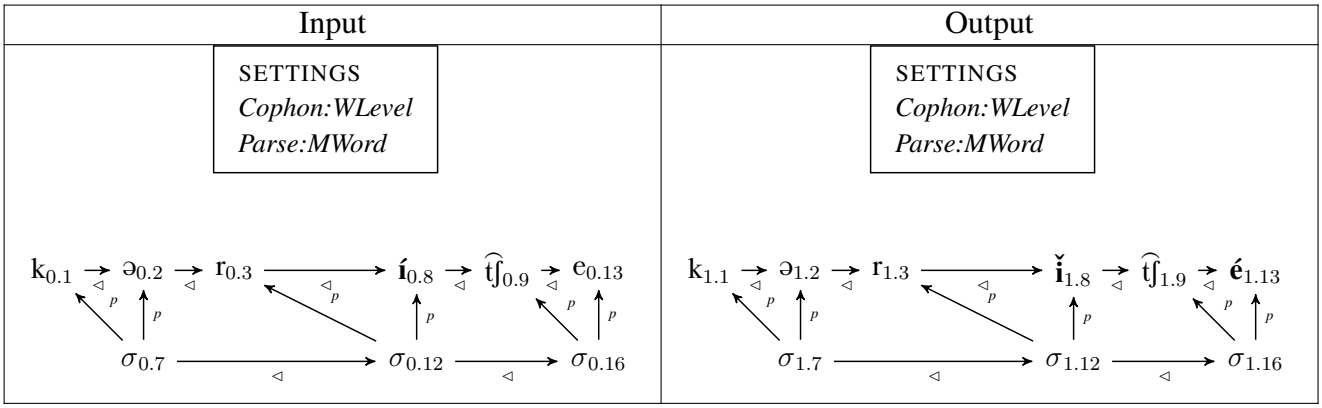
In this way, our logical notation directly captures the generation of prosodic structure from morphological structure.

5.4.4 Phonology: Word-level phonology blocks reduction

At the end of Cycle 3, we apply the Phonology component. As before, we examine the SETTINGS in order to determine what cophonology to apply. For *kər-ít͡ʃ-e*, the SETTINGS has the cophonol-

ogy label Cophon:WLevel. Thus, we should trigger the word-level cophonology of stress without reduction. We show the input and output below. We omit the morphology and operation list.

(49) *Phonology: Applying word-level phonology to an inflected word* kər-it͡ʃ-é



First off, we apply stress shift onto the new suffix *-e* by using the same stress shift functions from Cycle 1 (§5.2.6). They are triggered because the input satisfies the predicate for stress domains (50a). We then examine the SETTINGS to determine if reduction should apply. Because we are in the word-level, the predicate in (50b) is not satisfied so we do not trigger reduction.

(50) *Phonology: Predicates for phonological domains*

a. $\mathbf{StressDomain} \stackrel{\text{def}}{=} \text{Cophon:SLevel}(\text{SETTINGS}) \vee \text{Cophon:WLevel}(\text{SETTINGS})$

b. $\mathbf{ReductionDomain} \stackrel{\text{def}}{=} \text{Domain:Cophon:SLevel}(\text{SETTINGS})$

Readers can verify that functions used for stress shift (§5.2.6) all obey the cophonology condition **StressDomain**. Thus they will apply for the inflected word *kər-it͡ʃ-é*. In contrast, the reduction functions from Cycle 2 (§5.3.3) reference the condition **ReductionDomain** and are blocked.

6 Evaluating the cyclic architecture

The previous section showed how we developed a two-level declarative system for formalizing cyclic phonology. In this section, we explore the benefits and insights gained from using 2-level DP. These concern the breadth of empirical coverage of 2-level DP (§6.1), how 2-level DP provides metrics for analyzing the computational complexity or generative capacity of morphophonology (§6.2), and how 2-level DP can capture the cyclicity of lexical phonology (§6.3).

6.1 Breath of coverage

Using 2-level DP, this article formalized a small fragment of Armenian morphology and phonology. This fragment showed the use of logic to define, generate, and modify different types of structure (morphological, prosodic, and phonological). The logic also allowed us to define domains for phonological processes which differ based on their morphological trigger. This small fragment is part of a larger (though incomplete) 2-level DP grammar that is defined in Dolatian (2020a). We show in (51) the breadth of empirical phenomena that was formalized in Dolatian (2020a).

(51) *Aspects of the morphology-phonology interface which are defined*

Morphology	**Prosody**	**Phonological Rule Domains**
Affix linearization	Syllabification	Domains triggered by
Zero	Generating syllables	Morphemes
Prefix	Syllable ordering	Morphological constituents
Suffix	Resyllabification	Prosodic constituents
Mobile Affix	Tiers over syllables	
Affix Allomorphy	Mapping	
Inwards-sensitive	Generating prosodic constituents	
Outwards-sensitive	Misaligning prosodic constituents	
Phono-conditioned	Restructuring prosodic constituents	
Morpho-conditioned	Recursive Prosody	
Tiers over dominance	Generating recursion	
Compounding	Flattening recursion	
Formation	Compound prosody	
Head-marking		

The topics in the above table were formalized by using Armenian as a case study. In terms of morphology, Dolatian (2020a) formalizes the generation of compound structure. Various types of concatenative affixation processes are formalized, including rare patterns such as mobile affixation (Noyer 1994; Kim 2010; Bezrukov and Dolatian 2020). He provides logical definitions for various types of allomorphy, classified in terms of their trigger (phonological or morphological) and direction (inward or outward). By doing so, he provides exact computational definitions for hypotheses that allomorphy is local or has restricted directionality (Embick 2010). As for lexical prosody, he formalizes a wide array of prosodic configurations, including recursive prosody (Ito and Mester 2009), prosodic misalignment (McCarthy and Prince 1993), and compound prosody (Peperkamp 1997; Dolatian 2020c).

The utility of 2-level DP relies on its flexibility. We can formalize multiple simultaneous hidden structures, such as a morphological and prosodic trees. The trees can interact and set up contexts for diverse morphophonological processes. Thus, the takeaway is that the utility of 2-level DP is not restricted to small phonological fragments. Instead, 2-level DP can be used for a wide and nearly-comprehensive mathematical system for phonology and morphology.

6.2 Locality of phonology and morphology

The previous section showed that one property of 2-level DP is its *representational flexibility*. With 2-level DP, one can formalize many if not all aspects of phonology and morphology while maintaining fidelity to linguistic theory. Additionally, 2-level DP is *explicit*. Because it is both flexible and explicit, we can use the logical formalization to evaluate the generative capacity or computational complexity of different phonological and morphological processes (Rogers and Pullum 2011).

When defined over string-to-string transductions, morphology and phonology are largely finite-state definable. As langauges, both are regular and definable with FSAs; and as functions, both are rational relations and definable with 1-way FSTs (Kaplan and Kay 1994; Roark and Sproat 2007).[14] However, most phonological and morphological phenomena don't utilize the full power of regular languages or functions. Instead, most of them can be characterized by strict subclasses of 1-way FSAs and 1-way FSTs (Heinz 2018; Rogers et al. 2013; Heinz and Idsardi 2013), based on properties such as strict locality (Chandlee 2014; Chandlee and Heinz 2018; Chandlee et al. 2018; Chandlee 2017), strict piecewishood (Heinz 2010; Burness and McMullin 2020), tier-based strict locality (Heinz et al. 2011; Aksënova et al. 2016), and subsequentiality (Heinz and Lai 2013). These various subclasses are implementable (Aksënova 2020; Rogers and Lambert 2019a) and definable with formal logic (Heinz and Idsardi 2013; Lambert and Rogers 2019, 2020; Rogers and Lambert 2019b). The role of locality is salient not just in linear phonological processes but likewise in computational treatments of non-linear phonology (Jardine 2016, 2017a, 2020; Chandlee and Jardine 2019a; Rawski and Dolatian 2020; Dolatian and Rawski 2020).

One question is whether the shift from linear finite-state representations to tree-based logical transductions causes an increase in computational expressivity. Although such an increase exists, there are mathematically-defined metrics for evaluating the generative capacity of different types of logic-based systems. For example, First-Order (FO) logic is less expressive than Monadic Second-Order (MSO) logic (Filiot and Reynier 2016). Within 2-level DP, there is work on defining computationally local types of logical systems. Specifically, a logical function is computationally local if it does not *need* to use quantifiers, i.e., it is quantifier-free (QF) (Strother-Garcia 2018, 2019). Interestingly, most morphophonological processes are logically QF and local.

For this paper, the logical fragment in §5 was defined in terms of FO logic, with one area of MSO logic for syllable-ordering (§5.3.2). However, many of these FO or MSO transductions can be turned into QF functions by replacing binary relations with functions. For example, recall the hypothetical processes of word-initial voicing and word-final epenthesis from §3.2. We formalized these processes with the help of two FO predicates: $\mathbf{initial}(x)$ and $\mathbf{final}(x)$ (52a). These predicates are FO because they use quantifiers to check if there exists a segment y before or after x. These predicates use the binary relation of successor $\mathsf{succ}(x)$. However, we can replace the successor relation with two successor-based functions (52b). One function $\mathsf{F_L}{:}\mathsf{succ}(x)$ returns the variable y (the successor) such that $\mathsf{succ}(x, y)$ is true, while another function $\mathsf{F_R}{:}\mathsf{succ}(y)$ returns the variable x (the predecessor) such that $\mathsf{succ}(x, y)$ is true. By using these functions instead of

[14] The exception is total reduplication; see footnote 1.

relations, the predicates no longer need quantifiers and are now quantifier-free.

(52) a. *First-order predicates for initial and final segments*
$\mathbf{initial}(x) \stackrel{\text{def}}{=} \mathsf{segment}(x) \wedge \neq \exists y[\mathsf{segment}(y) \wedge \mathsf{succ}(y, x)]$
$\mathbf{final}(x) \stackrel{\text{def}}{=} \mathsf{segment}(x) \wedge \neq \exists y[\mathsf{segment}(y) \wedge \mathsf{succ}(x, y)]$
b. *Quantifier-free predicates for initial and final segments*
$\mathbf{initial}(x) \stackrel{\text{def}}{=} \mathsf{segment}(x) \wedge \mathsf{F_R}{:}\mathsf{succ}(x) \neq \text{NULL}$
$\mathbf{final}(x) \stackrel{\text{def}}{=} \mathsf{segment}(x) \wedge \mathsf{F_L}{:}\mathsf{succ}(x) \neq \text{NULL}$

In Dolatian (2020a), most of the logical formulas are written in FO logic because FO statements are easier to read. He provides a schema for converting binary relations into functions. He provides QF versions for the majority of logical formulas. With locality operationalized and equated with QF logic, Dolatian shows that most morphological and processes are computationally local even with the use of morphological and prosodic trees. However, the following areas are non-local and need FO logic:

(53) *Non-local areas of phonology and morphology*
a. mobile affixation
b. long-distance allomorphy
c. morphologically-triggered phonology
d. post-cyclic prosody.

Mobile affixation is non-local because the affixation references both the left and right edges of the input. Some types of allomorphy can likewise be non-local if the allomorph and the trigger are on opposite sides of the word. Morphologically-triggered cophonologies are non-local because the trigger for the phonological process is a potentially distant node in the morphological tree. For example, recall that the trigger for stem- vs. word-level cophonologies is the topmost morphological node. As long as there is no bound between this morphological node and the target phonological segment, then cophonologies can be computationally non-local.

However, the locality of the above processes can be analysis-dependent. For cophonologies, morpheme-based reanalyses of phonological processes (Pater 2007) can turn non-local processes into local ones. As for the prosody, if prosodic parsing is processed in a cyclic manner, then the computation is local. However, if the prosody is post-cyclic, i.e., all prosodic constituents are generated once at the end the derivation, then the prosody is non-local. The non-locality is however restricted to compound prosody.

In sum, the shift to richer representations for morphology and phonology shows that these linguistic phenomena are still largely computationally local and restrictive. But even when some areas are shown to be non-locally computed, these results are still enlightening on the computational restrictions and expressivity of morphophonological systems.

6.3 Computation of cyclicity

In our fragment, we use 2-level DP to not only encode the input-output relationships in morphophonology, but to also apply morphophonological processes in a cyclic manner. That is, the output of one round of morphophonology acts as the input to another. This section discusses the primary computational issue of cyclicity as the unbounded nature of word-formation rules.

In linguistics, cyclicity is a common aspect of generative phonology and morphology (Chomsky and Halle 1968; Brame 1974; Kiparsky 1982). The combination of cyclicity and interactionism flourished in early work in Lexical Phonology (Kiparsky 1982; Kaisse and Shaw 1985), continued into Stratal OT (Kiparsky 2015; Bermúdez-Otero 2018), and into contemporary phase-based approaches (Marvin 2002; Newell 2008; Embick 2010; Samuels 2011).

But in computational linguistics, cyclicity has been a challenging problem to formalize (Sproat 1992), with few implementations (Williams 1993) and few learnability results (Nazarov and Pater 2017). The same problem exists in computational syntax (Levelt 1974). The reason is because, in order for a phonological rule to be finite-state definable, the rule cannot apply to the locus of its structural change (Johnson 1972), i.e., to the same location in the output. Otherwise, the unbounded application of a phonological rule can create non-regular langauges. For example, a rule like $\emptyset \rightarrow ab / a\_b$ is regular, local, and logically QF. But the unbounded application of this rule can create the non-regular, context-free language $a^n b^n$.

It is this unboundedness which can cause a blow-up in expressivity. Unbounded rule application can simulate a Turing machine (Coleman 1995, 1998:77ff) or be computationally undecidable (Ristad 1990). Cyclic phonological rules are *a priori* supposed to apply to their own output, and thus we appear to reach an impasse. Kaplan and Kay (1994:365) put it nicely as: "In the worst case, in fact, we know that the computations of an arbitrary Turing machine can be simulated by a rewriting grammar with unrestricted rule reapplication."

This issue of unboundedness can be expressed succinctly as follows. It is not the case that there exists a number k such that for all words w ($\exists k, \forall w$), we can apply at most k cycles. In response to this problem, some formalizations of lexical phonology abandoned cyclicity and used context-free grammars (Coleman 1995; Cole and Coleman 1992), effectively treating the morphophonological module of grammar as non-regular. But this treatment is empirically and computationally unmotivated because of the near-ubiquity of regularity in morphophonology (Beesley and Karttunen 2003; Roark and Sproat 2007; Chandlee 2017).

Another approach is to place a bound on the number of cycles (Peters and Ritchie 1973). To our knowledge, this alternative has not been seriously developed because no a priori bound is clear. But on the other hand, it is the case that for every word w, there exists a number k of cycles which is *specific* to that word w ($\forall w, \exists k$), such that we apply exactly k cyclic. So for a given word w, there is a bound k in *run-time* or in *practice*. Given w, this allows us to use a finite-state grammar that can generate words with up to k cycles.[15] This approach prioritizes practical considerations

[15]This is similar to some treatments of total reduplication (Walther 2000; Beesley and Karttunen 2003) which push

over theoretical ones to side-step the question of the generative capacity of morphophonology. A related theoretical position one could take however would be to factor the morphophonological grammar into a regular morphophonological component and a control structure which allows the regular component to be called unboundedly many times. This is essentially the approach taken in Williams (1993) who uses a looping mechanism over the morphology and phonology. This practical approach recognizes the fact that phonological rules are regular per application (or up to k cycles) but that the generative capacity of the whole system can in principle be beyond regular.

A third approach is to identify restrictions on cyclic morphophonological rules so that the transitive closure of their composition remains regular. Such restrictions include the one that Johnson and Kaplan and Kay (JKK) recognized: phonological rules cannot apply to the locus of their structural change. This approach generates two questions: one empirical, and one formal. This approach looks the most promising.

The empirical question is whether *all* attested morphophonological rules obey the JKK restriction. In other words, are there cyclic morphophonological processes like $\emptyset \rightarrow ab / a\_b$ whose transitive closure is non-regular? Research around this question has consistently shown no. Although unbounded suffixation (a regular language) exists, there are no cases of unbounded circumfixation (a non-regular $a^n b^n$ language) (Aksënova et al. 2016). Although the semantics of morphological structures is non-regular because of center-embedding in tree structures (Langendoen 1981; Carden 1983; Oseki 2018; Oseki et al. 2019; Oseki and Marantz 2020), their surface morphotactics (Hammond 1993) and cyclic phonology are regular (Bjorkman and Dunbar 2016). The marked absence of non-regular morphotactics has been taken as evidence that, even if morphosemantics can in principle be non-regular, it is filtered through a finite-state morphophonological processing system (Hammond 1993).

It thus seems that that restriction on regularity is empirically robust. This leads to the formal question on how we can translate the formalization of the JKK restriction from string-to-string relations to logical transductions over nonlinear representations. Answering this question appears to us to be the most appealing at present.

7 Conclusion

This paper showcased how to construct a formal grammar with two-level Declarative Phonology. Our goal was to use 2-level DP to define a substantial fragment of the cyclic phonology of one language, Armenian. By doing so, we showed that cyclic processes are computationally definable. Our logical formalism was enriched enough to represent a wide array of morphophonological phenomena that are couched within cyclic phonology. By using an explicit system, we are able to determine the generative capacity of morphophonological processes. At least for our case study, we found that the brunt of phenomena are computationally local. A small number of well-defined

the work of total reduplication into a run-time process. This treatment is required if we try to formalize reduplication with 1-way FSTs, but not with 2-way FSTs (cf. for a review, see Dolatian and Heinz 2020).

areas are however not computationally local.

References

Aksënova, A. (2020). *Tool-assisted induction of subregular languages and mappings*. Ph. D. thesis, Stony Brook University.

Aksënova, A., T. Graf, and S. Moradi (2016). Morphotactics as tier-based strictly local dependencies. In *Proceedings of the 14th sigmorphon workshop on computational research in phonetics, phonology, and morphology*, pp. 121–130.

Beesley, K. and L. Karttunen (2003). *Finite-state morphology: Xerox tools and techniques*. Stanford, CA: CSLI Publications.

Bermúdez-Otero, R. (2011). Cyclicity. In M. van Oostendorp, C. Ewen, E. Hume, and K. Rice (Eds.), *The Blackwell companion to phonology*, Volume 4, pp. 2019–2048. Malden, MA: Wiley-Blackwell.

Bermúdez-Otero, R. (2018). Stratal phonology. In S. Hannahs and A. R. K. Bosch (Eds.), *The Routledge handbook of phonological theory*, pp. 382–410. Abingdon: Routledge.

Bezrukov, N. and H. Dolatian (2020). Mobile affixes across Western Armenian: Conflicts across modules. In *University of Pennsylvania Working Papers in Linguistics*, Volume 26.

Bhaskar, S., J. Chandlee, A. Jardine, and C. Oakden (2020). Boolean monadic recursive schemes as a logical characterization of the subsequential functions. In A. Leporati, C. Martín-Vide, D. Shapira, and C. Zandron (Eds.), *Proceedings of the 14th International Conference on Language and Automata Theory and Applications (LATA 2020)*.

Bird, S. (1995). *Computational phonology: A constraint-based approach*. Studies in Natural Language Processing. Cambridge: Cambridge University Press.

Bird, S., J. S. Coleman, J. Pierrehumbert, and J. M. Scobbie (1992). Declarative phonology. In *Proceedings of the XVth International Congress of Linguistics*, University Laval, Quebec.

Bird, S. and T. M. Ellison (1994). One-level phonology: Autosegmental representations and rules as finite automata. *Computational Linguistics 20*(1), 55–90.

Bird, S. and E. Klein (1994). Phonological analysis in typed feature systems. *Computational linguistics 20*(3), 455–491.

Bjorkman, B. and E. Dunbar (2016). Finite-state phonology predicts a typological gap in cyclic stress assignment. *Linguistic Inquiry 47*(2), 351–363.

Booij, G. and R. Lieber (1993). On the simultaneity of morphological and prosodic structure. In S. Hargus and E. M. Kaisse (Eds.), *Studies in lexical phonology*, Volume 4 of *Phonetics and Phonology*, pp. 23–44. San Diego: Academic Press.

Brame, M. K. (1974). The cycle in phonology: Stress in Palestinian, Maltese, and Spanish. *Linguistic Inquiry 5*(1), 39–60.

Büchi, J. R. (1960). Weak second-order arithmetic and finite automata. *Mathematical Logic Quarterly 6*(1-6), 66–92.

Burness, P. A. and K. McMullin (2020). Modelling non-local maps as strictly piecewise functions. In *Proceedings of the Society for Computation in Linguistics*, Volume 3.

Carden, G. (1983). The non-finite-state-ness of the word formation component. *Linguistic Inquiry 14*(3), 537–541.

Chandlee, J. (2014). *Strictly Local Phonological Processes*. Ph. D. thesis, University of Delaware, Newark, DE.

Chandlee, J. (2017). Computational locality in morphological maps. *Morphology 27*(4), 1–43.

Chandlee, J., R. Eyraud, J. Heinz, A. Jardine, and J. Rawski (2019, 18–19 July). Learning with partially ordered representations. In *Proceedings of the 16th Meeting on the Mathematics of Language*, Toronto, Canada, pp. 91–101. Association for Computational Linguistics.

Chandlee, J. and J. Heinz (2018). Strict locality and phonological maps. *Linguistic Inquiry 49*(1), 23–60.

Chandlee, J., J. Heinz, and A. Jardine (2018). Input strictly local opaque maps. *Phonology 35*(2), 171–205.

Chandlee, J. and A. Jardine (2019a). Autosegmental input strictly local functions. *Transactions of the Association for Computational Linguistics 7*, 157–168.

Chandlee, J. and A. Jardine (2019b). Quantifier-free least fixed point functions for phonology. In *Proceedings of the 16th Meeting on the Mathematics of Language (MoL 16)*, Toronto, Canada. Association for Computational Linguistics.

Chew, P. (2003). *A computational phonology of Russian*. Parkland, FL: Universal-Publishers.

Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on information theory 2*(3), 113–124.

Chomsky, N. and M. Halle (1968). *The sound pattern of English.* Cambridge, MA: MIT Press.

Chomsky, N., M. Halle, and F. Lukoff (1956). On accent and juncture in English. In *For Roman Jakobson*, pp. 65–80. The Hague: Mouton.

Church, K. W. (1983). *Phrase-structure parsing: A method for taking advantage of allophonic constraints*. Ph. D. thesis, Massachusetts Institute of Technology.

Cole, J. (1995). The cycle in phonology. In J. Goldsmith (Ed.), *The Handbook of Phonological Theory* (1 ed.)., pp. 70–113. Cambridge, MA: Blackwell Publishers.

Cole, J. and J. Coleman (1992). No need for cyclicity in generative phonology. In C. P. Canakis, G. P. Chan, and J. M. Denton (Eds.), *Proceedings of the 28th Regional Meeting of the Chicago Linguistics Society*, Volume 2, Chicago, IL, pp. 36–50. University of Chicago.

Coleman, J. (1995). Declarative lexical phonology. In J. Durand and F. Katamba (Eds.), *Frontiers of phonology: Atoms, structures, derivations*, pp. 333–383. London: Longman.

Coleman, J. (1998). *Phonological representations: Their names, forms and powers*. Cambridge: Cambridge University Press.

Courcelle, B. (1994). Monadic second-order definable graph transductions: A survey. *Theoretical Computer Science 126*(1), 53–75.

Courcelle, B. (1997). The expression of graph properties and graph transformations in monadic second-order logic. In G. Rozenberg (Ed.), *Handbook of Graph Grammars and Computing by Graph Transformations*, Volume 1, pp. 313–400. World Scientific.

Courcelle, B. and J. Engelfriet (2012). *Graph Structure and Monadic Second-Order Logic, a Language Theoretic Approach*. Cambridge: Cambridge University Press.

Culy, C. (1985). The complexity of the vocabulary of Bambara. *Linguistics and Philosophy 8*, 345–351.

Danis, N. and A. Jardine (2019). Q-theory representations are logically equivalent to autosegmental representations. In *Proceedings of the Society for Computation in Linguistics*, Volume 2, pp. 29–38.

Dolatian, H. (2020a). *Computational locality of cyclic phonology in Armenian*. Ph. D. thesis, Stony Brook University.

Dolatian, H. (2020b). Cyclicity and prosodic misalignment in Armenian stems: Interaction of morphological and prosodic cophonologies. *Natural Language and Linguistic Theory*.

Dolatian, H. (2020c). The role of heads and cyclicity in bracketing paradoxes in Armenian compounds. *Morphology*.

Dolatian, H. and J. Heinz (2020). Computing and classifying reduplication with 2-way finite-state transducers. *Journal of Language Modeling 8*, 79–250.

Dolatian, H., J. Heinz, and K. Strother-Garcia (Eds.) (in prep). *Doing Computational Phonology*. Oxford: Oxford University Press.

Dolatian, H. and J. Rawski (2020). Multi input strictly local functions for templatic morphology. In *Proceedings of the Society for Computation in Linguistics*, Volume 3.

Downing, L. J. (1999). Prosodic stem $\neq$ prosodic word in Bantu. In T. A. Hall and U. Kleinhenz (Eds.), *Studies on the phonological word*, Volume 174, pp. 73–98. Amsterdam/Philadelphia: John Benjamins Publishing.

Ellison, T. M. and J. Scobbie (Eds.) (1993). *Computational Phonology*. University of Edinburgh: Centre for Cognitive Science.

Embick, D. (2010). *Localism versus globalism in morphology and phonology*, Volume 60 of *Linguistic Inquiry Monographs*. Cambridge, MA: MIT Press.

Enderton, H. (2001). *A mathematical introduction to logic*. Academic Press.

Engelfriet, J. and H. J. Hoogeboom (2001, April). MSO definable string transductions and two-way finite-state transducers. *Transactions of the Association for Computational Linguistics 2*(2), 216–254.

Filiot, E. and P.-A. Reynier (2016, August). Transducers, logic and algebra for functions of finite words. *ACM SIGLOG News 3*(3), 4–19.

Gorman, K. (2016). Pynini: A python library for weighted finite-state grammar compilation. In *Proceedings of the SIGFSM Workshop on Statistical NLP and Weighted Automata*, Berlin, Germany, pp. 75–80. Association for Computational Linguistics.

Graf, T. (2010). Logics of phonological reasoning. Master's thesis, University of California, Los Angeles.

Hammond, M. (1993). On the absence of category-changing prefixes in English. *Linguistic Inquiry 24*(3), 562–567.

Heinz, J. (2010). Learning long-distance phonotactics. *Linguistic Inquiry 41*(4), 623–661.

Heinz, J. (2018). The computational nature of phonological generalizations. In L. Hyman and F. Plank (Eds.), *Phonological Typology*, Phonetics and Phonology, Chapter 5, pp. 126–195. Berlin: Mouton de Gruyter.

Heinz, J., C. de la Higuera, and M. van Zaanen (2015). *Grammatical Inference for Computational Linguistics*. Synthesis Lectures on Human Language Technologies. Morgan and Claypool.

Heinz, J. and W. Idsardi (2013). What complexity differences reveal about domains in language. *Topics in Cognitive Science 5*(1), 111–131.

Heinz, J. and R. Lai (2013). Vowel harmony and subsequentiality. In A. Kornai and M. Kuhlmann (Eds.), *Proceedings of the 13th Meeting on the Mathematics of Language (MoL 13)*, Sofia, Bulgaria, pp. 52–63. Association for Computational Linguistics.

Heinz, J., C. Rawal, and H. G. Tanner (2011). Tier-based strictly local constraints for phonology. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Short papers-Volume 2*, pp. 58–64. Association for Computational Linguistics.

Hulden, M. (2006). Finite-state syllabification. In A. Yli-Jyrä, L. Karttunen, and J. Karhumäki (Eds.), *Finite-State Methods and Natural Language Processing. FSMNLP 2005. Lecture Notes in Computer Science*, Volume 4002. Berlin/Heidelberg: Springer.

Hulden, M. (2009). Foma: A finite-state compiler and library. In *Proceedings of the Demonstrations Session at EACL 2009*, Athens, Greece, pp. 29–32. Association for Computational Linguistics.

Idsardi, W. J. (2009). Calculating metrical structure. In E. Raimy and C. E. Cairns (Eds.), *Contemporary views on architecture and representations in phonology*, Number 48 in Current Studies in Linguistics, pp. 191–211. Cambridge, MA: MIT Press.

Inkelas, S. (2014). *The interplay of morphology and phonology*. Oxford: Oxford University Press.

Ito, J. and A. Mester (2009). The extended prosodic word. In J. Grijzenhout and B. Kabak (Eds.), *Phonological domains: Universals and deviations*, Number 16 in Interface explorations, pp. 135–194. Berlin: Mouton de Gruyter.

Jardine, A. (2016). *Locality and non-linear representations in tonal phonology*. Ph. D. thesis, University of Delaware, Newark, DE.

Jardine, A. (2017a). The local nature of tone-association patterns. *Phonology 34*(2), 363–384.

Jardine, A. (2017b). On the logical complexity of autosegmental representations. In *Proceedings of the 15th Meeting on the Mathematics of Language*, pp. 22–35.

Jardine, A. (2020). Melody learning and long-distance phonotactics in tone. *Natural Language & Linguistic Theory 38*, 1145–1195.

Jardine, A., N. Danis, and L. Iacoponi (2020). A formal investigation of Q-theory in comparison to autosegmental representations. *Linguistic Inquiry*.

Johnson, C. D. (1972). *Formal aspects of phonological description*. The Hague: Mouton.

Kaisse, E. M. and P. A. Shaw (1985). On the theory of lexical phonology. *Phonology 2*(1), 1–30.

Kaplan, R. M. and M. Kay (1994). Regular models of phonological rule systems. *Computational linguistics 20*(3), 331–378.

Kim, Y. (2010). Phonological and morphological conditions on affix order in Huave. *Morphology 20*(1), 133–163.

Kiparsky, P. (1982). Lexical morphology and phonology. In I.-S. Yang (Ed.), *Linguistics in the morning calm: Selected papers from SICOL-1981*, pp. 3–91. Seoul: Hansin.

Kiparsky, P. (2015). Stratal OT: A synopsis and FAQs. In Y. E. Hsiao and L.-H. Wee (Eds.), *Capturing phonological shades within and across languages*, pp. 2–44. Cambridge: Cambridge Scholars Publishing.

Koser, N. and A. Jardine (2019). The computational nature of stress assignment. In *Proceedings of AMP 2019*.

Koser, N., C. Oakden, and A. Jardine (2019). Tone association and output locality in non-linear structures. In *Supplemental proceedings of AMP 2019*.

Koskenniemi, K. (1983). *Two-level morphology: A General Computational Model for Word-Form Recognition and Production*. Ph. D. thesis, University of Helsinki.

Lambert, D. and J. Rogers (2019). A logical and computational methodology for exploring systems of phonotactic constraints. In *Proceedings of the Society for Computation in Linguistics*, Volume 2, pp. 247–256.

Lambert, D. and J. Rogers (2020). Tier-based strictly local stringsets: Perspectives from model and automata theory. In *Proceedings of the Society for Computation in Linguistics*, Volume 3.

Langendoen, D. T. (1981). The generative capacity of word-formation components. *Linguistic Inquiry 12*(2), 320–322.

Levelt, W. (1974). *Formal grammars in linguistics and psycholinguistics*. The Hague: Mouton.

Libkin, L. (2013). *Elements of finite model theory*. Springer Science & Business Media.

Marantz, A. (2007). Phases and words. In S.-H. Choe, D.-W. Yang, Y.-S. Kim, S.-H. Kim, and A. Marantz (Eds.), *Phases in the theory of grammar*, pp. 191–222. Seoul: Dong-In Publishing Co.

Marvin, T. (2002). *Topics in the stress and syntax of words*. Ph. D. thesis, Massachusetts Institute of Technology.

McCarthy, J. J. and A. Prince (1993). Generalized alignment. In G. Booij and J. van Marie (Eds.), *Yearbook of morphology 1993*, pp. 79–153. Dordrecht: Kluwer Academic Publishers.

Miller, G. A. and N. Chomsky (1963). Finitary models of language users. In R. D. Luce, R. R. Bush, and E. Galanter (Eds.), *Handbook of Mathematical Psychology*, Volume II, pp. 419–491. New York: John Wiley.

Nazarov, A. and J. Pater (2017). Learning opacity in stratal maximum entropy grammar. *Phonology 34*(2), 299–324.

Nespor, M. and I. Vogel (1986). *Prosodic phonology*. Dordrecht: Foris.

Newell, H. (2008). *Aspects of the morphology and phonology of phases*. Ph. D. thesis, McGill University, Montreal, QC.

Noyer, R. (1994). Mobile affixes in Huave: Optimality and morphological wellformedness. In E. Duncan, D. Farkas, and P. Spaelti (Eds.), *Proceedings of the Twelfth West Coast Conference on Formal Linguistics. Stanford: CSLI*, Stanford, pp. 67–82. CSLI.

Oakden, C. (2020). Notational equivalence in tonal geometry. *Phonology 37*, 257–296.

Ogden, R. (1999). A declarative account of strong and weak auxiliaries in English. *Phonology 16*(1), 55–92.

Oseki, Y. (2018). *Syntactic structures in morphological processing*. Ph. D. thesis, New York University.

Oseki, Y. and A. Marantz (2020). Modeling human morphological competence. *Frontiers in Psychology 11*.

Oseki, Y., C. Yang, and A. Marantz (2019). Modeling hierarchical syntactic structures in morphological processing. In *Proceedings of the Workshop on Cognitive Modeling and Computational Linguistics*, Minneapolis, Minnesota, pp. 43–52. Association for Computational Linguistics.

Pater, J. (2007). The locus of exceptionality: Morpheme-specific phonology as constraint indexation. In L. Bateman, M. O'Keefe, E. Reilly, and A. Werle (Eds.), *University of Massachusetts Occasional Papers in Linguistics 32: Papers in Optimality Theory III*, pp. 187–207. Amherst, MA: Graduate Linguistics Student Association, University of Massachusetts.

Pater, J. (2019). Generative linguistics and neural networks at 60: Foundation, friction, and fusion. *Language 95*(1), e41–e74.

Peperkamp, S. A. (1997). *Prosodic words*. The Hague: Holland Academic Press.

Peters, P. S. and R. W. Ritchie (1973). On the generative power of transformational grammars. *Information sciences 6*, 49–83.

Potts, C. and G. K. Pullum (2002). Model theory and the content of OT constraints. *Phonology 19*(3), 361–393.

Prince, A. and P. Smolensky (2004). *Optimality Theory: Constraint Interaction in Generative Grammar*. Oxford: Blackwell Publishing.

Rawski, J. (2019). Tensor product representations of subregular formal languages. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019) workshop on Neural-Symbolic Learning and Reasoning*.

Rawski, J. and H. Dolatian (2020). Multi input strictly local functions for tonal phonology. In *Proceedings of the Society for Computation in Linguistics*, Volume 3.

Ristad, E. S. (1990). *Computational structure of human language*. Ph. D. thesis, Massachusetts Institute of Technology.

Roark, B. and R. Sproat (2007). *Computational Approaches to Morphology and Syntax*. Oxford: Oxford University Press.

Roche, E. and Y. Schabes (Eds.) (1997). *Finite-state language processing*. Cambridge: MIT press.

Rogers, J. (1998). *A Descriptive Approach to Language-Theoretic Complexity*. Stanford, CA: CSLI Publications.

Rogers, J., J. Heinz, M. Fero, J. Hurst, D. Lambert, and S. Wibel (2013). Cognitive and sub-regular complexity. In G. Morrill and M.-J. Nederhof (Eds.), *Formal Grammar*, Volume 8036 of *Lecture Notes in Computer Science*, pp. 90–108. Springer.

Rogers, J. and D. Lambert (2019a). Extracting subregular constraints from regular stringsets. *Journal of Language Modelling 7*(2), 143–176.

Rogers, J. and D. Lambert (2019b). Some classes of sets of structures definable without quantifiers. In *Proceedings of the 16th Meeting on the Mathematics of Language*, Toronto, Canada, pp. 63–77. Association for Computational Linguistics.

Rogers, J. and G. Pullum (2011). Aural pattern recognition experiments and the subregular hierarchy. *Journal of Logic, Language and Information 20*, 329–342.

Samuels, B. D. (2011). *Phonological architecture: A biolinguistic perspective*. Oxford Studies in Biolinguistics. Oxford University Press.

Scheer, T. (2011). *A guide to morphosyntax-phonology interface theories: How extra-phonological information is treated in phonology since Trubetzkoy's Grenzsignale*. Berlin: Mouton de Gruyter.

Scobbie, J. M. (1991). *Attribute value phonology*. Ph. D. thesis, University of Edinburgh.

Scobbie, J. M. (1993). Constraint violation and conflict from the perspective of declarative phonology. *Canadian Journal of Linguistics/Revue canadienne de linguistique 38*(2), 155–167.

Scobbie, J. M., J. S. Coleman, and S. Bird (1996). Key aspects of declarative phonology. In J. Durand and B. Laks (Eds.), *Current Trends in Phonology: Models and Methods*, Volume 2. Salford, Manchester: European Studies Research Institute.

Selkirk, E. (1982). *The syntax of words*. Number 7 in Linguistic Inquiry Monographs. Cambridge, Mass: MIT Press.

Selkirk, E. (2011). The syntax-phonology interface. In J. Goldsmith, J. Riggle, and A. C. L. Yu (Eds.), *The Handbook of Phonological Theory* (2 ed.)., pp. 435–483. Oxford: Blackwell.

Sproat, R. W. (1992). *Morphology and computation*. Cambridge,MA: MIT press.

Strother-Garcia, K. (2018). Imdlawn Tashlhiyt Berber syllabification is quantifier-free. In *Proceedings of the Society for Computation in Linguistics*, Volume 1, pp. 145–153.

Strother-Garcia, K. (2019). *Using model theory in phonology: a novel characterization of syllable structure and syllabification*. Ph. D. thesis, University of Delaware.

Strother-Garcia, K., J. Heinz, and H. J. Hwangbo (2017). Using model theory for grammatical inference: A case study from phonology. In *International Conference on Grammatical Inference*, pp. 66–78.

Vaux, B. (1998). *The phonology of Armenian*. Oxford: Clarendon Press.

Vu, M. H., A. Zehfroosh, K. Strother-Garcia, M. Sebok, J. Heinz, and H. G. Tanner (2018). Statistical relational learning with unconventional string models. *Frontiers in Robotics and AI 5*, 76.

Walther, M. (2000). Finite-state reduplication in one-level prosodic morphology. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*, NAACL 2000, Seattle, Washington, pp. 296–302. Association for Computational Linguistics.

Wheeler, D. W. (1981). *Aspects of a categorial theory of phonology.* Ph. D. thesis, University of Massachusetts Amherst.

Williams, S. M. (1993). *LexPhon: A computational implementation of aspects of lexical phonology.* Ph. D. thesis, University of Reading.

Yu, K. (2017). Advantages of constituency: Computational perspectives on Samoan word prosody. In *International Conference on Formal Grammar 2017*, Berlin, pp. 105–124. Spring.

Yu, K. M. (2019). Parsing with minimalist grammars and prosodic trees. In R. C. Berwick and E. P. Stabler (Eds.), *Minimalist Parsing*, pp. 69–109. London: Oxford University Press.