2017

Computational Locality in Morphological Maps

Jane Chandlee
Haverford College, jchandlee@haverford.edu

Morphology (2017) 27:599–641
DOI 10.1007/s11525-017-9316-9

Computational locality in morphological maps

Jane Chandlee[1]

Received: 2 August 2016 / Accepted: 23 October 2017 / Published online: 7 November 2017

Abstract This paper presents a computational investigation of a range of morphological operations. These operations are first represented as *morphological maps*, or functions that take a stem as input and return an output with the operation applied (e.g., the *ing*-suffixation map takes the input 'dɹɪŋk' and returns 'dɹɪŋk+ɪŋ'). Given such representations, each operation can be classified in terms of the computational complexity needed to map a given input to its correct output. The set of operations analyzed includes various types of affixation, reduplication, and non-concatenative morphology. The results indicate that many of these operations require less than the power of regular relations (i.e., they are subregular functions), the exception being total reduplication. A comparison of the maps that fall into different complexity classes raises important questions for our overall understanding of the computational nature of phonology, morphology, and the morpho-phonological interface.

Keywords Morphological maps · Computational locality · Subregularity · Morpho-phonological interface

1 Introduction

Classifying natural language patterns in terms of their computational complexity—defined in this paper as the amount of computational power needed to recognize and/or generate the pattern—is one approach to understanding what kinds of patterns can and cannot exist in natural language. In addition, computational analyses of patterns in different linguistic domains offer one perspective on how these domains fundamentally differ (e.g., Bromberger and Halle 1989). In particular, previous work

✉ J. Chandlee
jchandlee@haverford.edu

[1] Haverford College, Tri-Co Department of Linguistics, 370 Lancaster Avenue, Haverford, PA 19041, USA

has shown that syntactic patterns exist which are context-free (Chomsky 1956) and context-sensitive (Shieber 1985; Kobele 2006), while virtually all phonological patterns are known to be regular relations (i.e., finite state) (Johnson 1972; Koskenniemi 1983; Kaplan and Kay 1994) with substantial evidence indicating that they are in fact properly *subregular* (Heinz 2009, 2010; Heinz et al. 2011; Chandlee et al. 2012; Gainor et al. 2012; Chandlee and Heinz 2012; Heinz and Lai 2013; Chandlee 2014; Luo 2013; Payne 2017), meaning they can be represented with proper subclasses of the regular relations. Collectively these findings suggest that syntax has the potential to be more computationally complex than phonology.

Perhaps not surprisingly, morphology falls somewhere in between, with varying claims that it is regular like phonology (Langendoen 1981) and context-free (Carden 1983) or even context-sensitive (Culy 1985) like syntax. Heinz and Idsardi (2013) conjecture that patterns classified as morpho-phonological will have similar computational properties as phonological patterns (i.e., be regular or subregular), and likewise morpho-syntactic patterns will be more computationally complex (i.e., be non-regular). The extent to which this hypothesis holds is unknown, and testing it fully is a large undertaking. The primary goal of this paper is to establish what is currently known about the computational nature of morphological operations and identify the significant open questions.

The previous work on natural language complexity (reviewed in detail below) has followed two approaches. One is to treat a pattern as a *formal language*, or a set of strings that obey a particular restriction or constraint. For example, a phonotactic constraint like *NK (i.e., nasal-stop sequences must be homorganic) can be represented as the set of strings that do *not* contain a violating NK sequence. The complexity of different formal languages can be compared in terms of the needed computational power of the grammar that generates them. A second approach is to analyze a pattern as a *map*, or a relation/function from one set of strings to another. As with formal languages, maps can also be categorized in terms of the complexity of the computation needed to correctly map an input string to its output string. The previous work on the complexity of morphology followed the first approach and analyzed the set of strings that the word formation component must be able to generate. This paper will take the second approach and define *morphological maps*, with the goal of characterizing the various kinds of operations that actually generate those strings.

As an example of what is meant by a morphological map, a suffixation map is shown in (1). This map takes an input string and returns it with the string ɪŋ appended to the end.

(1) f_{prog} (spik) = spik+ɪŋ

The goal is to identify the computational properties of such maps, properties that hold regardless of the theoretical formalism used to describe the map. As will be reviewed below, the same approach for studying phonological maps has led to significant results for the computational nature of the transformation from underlying to surface forms. Following Tesar (2008, 2014) and Baković (2013), the use of the term *map* here reflects the fact that such investigations aim to reveal properties of the transformation from underlying representation (UR) to surface representation (SR), properties that are independent of any particular grammatical formalism. In other words,

both rule-based formalisms like SPE (Chomsky and Halle 1968) and constraint-based formalisms like Optimality Theory (Prince and Smolensky 2004), Harmonic Grammar (Legendre et al. 1990), or Harmonic Serialism (McCarthy 2000; Pater 2012) assume the existence of a map from a UR to an SR, though they of course differ greatly in how that map is achieved.

The concept of a map like in (1) may be more consistent with some morphological theories than others. In particular (in the terms of Hockett 1954), the concept of a morphological map may be more in line with Item-and-Process or Word and Paradigm theories (Anderson 1992; Aronoff 1994; Stump 2001) than with Item-and-Arrangement theories (Halle and Marantz 1993). The objective here is not to provide evidence in favor of one theory over the other—indeed, Roark and Sproat (2007, Chap. 3) argue that computationally-speaking there is no difference between these approaches. Rather, the goal is to identify the computational properties of morphological operations under the assumption that they can be represented as maps. A comparable computational investigation under different assumptions may require a different methodology than the one employed here.

The key result of the analyses to follow is that morphological maps predominantly belong to well-defined and restricted *subregular* classes of functions. The operations to be analyzed include the following:

– (non-reduplicative) affixation
– partial reduplication
– total reduplication
– featural affixation
– truncation

A few comments about compounding and templatic morphology will also be offered, with more thorough analyses being left for future work.

It has already been established (see e.g., Roark and Sproat 2007; Beesley and Karttunen 2003; Hulden 2009a,b) that these operations are *regular relations*—meaning they can be modeled with finite state transducers, a formalism that will be introduced in Sect. 2—with the exception of total reduplication.[1] But it will be shown that—again with the exception of total reduplication—all of these operations can in fact be modeled with properly subregular classes of transducers. Thus morphology may in fact be less complex than has been previously assumed. The subregular nature of morphology is also significant from the perspective of learning, since—unlike the regular relations—the subregular functions used in the analyses to follow are provably and efficiently learnable from positive data (Oncina et al. 1993; Chandlee et al. 2014, 2015; Jardine et al. 2014).

Another note on how the results that will be presented in this paper compare to the previous work on computational complexity and natural language. The majority of that work (which will be reviewed in Sect. 3 below) applied the following type of argument: language domain X does not belong to complexity class Y because there exists at least one example of a X pattern that cannot be classified in Y. For example, syntax is not *context-free* because serial verb case-marking in Swiss German is

[1] Though finite-state approximations of total reduplication have been proposed and implemented by several of these authors.

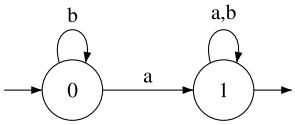


Fig. 1 Finite state acceptor for $\mathbb{L}_1$

context-sensitive (Shieber 1985).[2] In contrast, the primary objective of the current paper is not to propose a new upper bound on the complexity of morphology *as a whole*. Rather, a catalog of various morphological operations will be analyzed individually, as a means of getting a more nuanced view of the nature of the computations involved in word formation.

The paper is structured as follows. Section 2 presents the requisite background for understanding the computational results presented in the paper (i.e., what it means for a pattern to be regular, subregular, etc.). Section 3 surveys the key previous results on the computational complexity of syntactic, morphological, and phonological patterns. Section 4 presents the computational analyses of a set of morphological operations, including (non-reduplicative) affixation (Sect. 4.1), partial and total reduplication (Sects. 4.2–4.4), featural affixation (Sect. 4.5), and truncation (Sect. 4.6). Section 5 discusses the significance of these results and addresses important remaining questions. Section 6 concludes.

2 Computational background

The complexity classes used to classify language patterns in the previous and current work come from theoretical computer science, in particular from formal language theory. A formal language begins with a finite set of symbols called an *alphabet*; this set is typically designated with Σ. A *string* or *word* is formed by concatenating symbols from Σ together, and Σ^* designates the infinite set of all such strings. A *language* is then a subset of Σ^*. For example, if Σ is the set {a, b}, then Σ^* is the infinite set of strings of a's and b's of any length, and we can define a language $\mathbb{L}_1$ in which all words have at least one 'a': $\mathbb{L}_1 = \{\text{a, aa, ba, ab, } \ldots, \text{ bbbbbbbba, } \ldots\}$.

The types of symbols included in Σ depends on the type of language pattern being analyzed. Σ may include words or syntactic categories for an analysis of syntax, morphemes for morphology, or phonemes and allophones for phonology. The strings of the formal language in these cases would be permissible sentences, words, or underlying/surface forms, respectively.

A formal language is classified as *regular* if it can be represented with a finite state acceptor (FSA).[3] For example, the FSA in Fig. 1 is a representation of the language $\mathbb{L}_1$ defined above.

A FSA is a set of states (in Fig. 1 the states are labeled 0 and 1) and a set of labeled transitions between states. Starting in a designated start state (marked with an

[2] These terms will be explained in the sections to follow.

[3] There are other definitions of regular languages based in other formalisms (e.g., regular expressions, monadic second order logic, etc.), but this paper will use only automata-theoretic characterizations throughout.

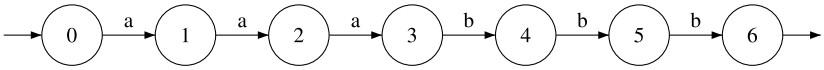


Fig. 2 FSA that recognizes the string 'aaabbb'

unlabeled incoming arrow, as in state 0 in Fig. 1), a given string is read one symbol at a time and transitions are followed according to the current symbol being read. If at the end of the string the FSA is in an accepting state (marked with an outgoing arrow with no destination state, as in state 1), then the string is in the language that the FSA represents. If the FSA ends in a non-accepting state, the string is not in the language. It is easy to see in the figure that as soon as an 'a' is read, the FSA proceeds to the accepting state 1, where it remains until the end of the string is reached. If no 'a' is ever read (i.e., the string contains only b's), then the FSA never leaves state 0. Since 0 is not an accepting state, the string will be rejected. Thus the FSA will correctly accept all and only those strings with at least one 'a'.

A language that is non-regular cannot be represented with a FSA, because whatever information is needed to distinguish strings that are and are not in the language requires an infinite number of states. As an example, consider again $\Sigma = \{\mathrm{a}, \mathrm{b}\}$ and the language $\mathbb{L}_n$ that includes strings of the form $\mathrm{a}^n\mathrm{b}^n$, where n is any integer (i.e., strings starting with any number n of a's followed by the same number of b's, so 'ab', 'aabb', 'aaabbb', 'aaaabbbb', etc.). What would a FSA that recognizes this language look like? It would have to first identify how many a's the string begins with and then verify that an identical number of b's follows. So if $n = 3$, the FSA in Fig. 2 would recognize that 'aaabbb' is in the language.

The problem is that since n can be any of the infinite set of integers, the complete FSA would need an infinite number of branches like Fig. 2, one for each possible integer. By definition a FSA can only have a finite number of states; therefore this language cannot be represented with a FSA and is therefore not a regular language.[4] In this way the finite state formalism serves as a classification tool. The statements 'language X is regular' and 'language X can be described/represented/modeled with a FSA' are equivalent. Likewise, the statements 'language X is not regular' and 'language X cannot be described/represented/modeled with any FSA' are also equivalent.

The finite state formalism can also be used to analyze string-to-string maps. Just as a FSA can represent a formal language, a finite state *transducer* (FST) can represent a function/relation/map. The difference between FSAs and FSTs is that the transition labels of FSTs include both an input symbol and an output string. As an input string is read by a FST, it produces an output string by concatenating the output strings of all the transitions it follows through the states. An example FST is shown in Fig. 3. Like the FSA in Fig. 1, the FST reads strings of a's and b's. The first time it reads an 'a', it outputs a 'b'. All other a's are outputted as a's and all b's are outputted as b's. This FST represents an infinite map that includes string pairs like ('aaa', 'baa'), ('bba', 'bbb'), ('aba', 'bba'), etc.

[4]This is not a proof that the language is non-regular, just an intuitive explanation. To see how an actual proof can be constructed, readers are referred to Hopcroft et al. (2000).

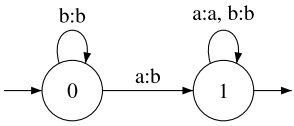


Fig. 3 FST that maps strings from {a, b}* to {a, b}*

This paper will make extensive use of the FST formalism as a means of *classifying* various types of morphological maps in terms of their computational complexity.[5] It is important to note, however, that this is certainly not the first or only application of finite state representations of morphology. The primary application thus far has been *morphological analysis*, in which a FST representation of a language's morpho-phonological system is constructed to be used for both generation and recognition (Beesley and Karttunen 2003; Hulden 2009a,b).

As an example, we can build such a system by starting with a set of lexical items augmented with tags for part of speech, tense, number, person, etc. For English this includes items like *run+V+3P+Sg, run+N+Pl*, etc. These tagged lexical items are used as input to a FST that replaces each tag with its corresponding affix (or deletes the tag if no affix is used to express it):[6]

(2) a. run+V+3P+Sg ↦ runs
b. run+N+Pl ↦ runs
c. stretch+V+3P+Sg ↦ stretchs

Additional FSTs take care of any phonological processes that are triggered by the affixation. For example, in English *e*-insertion is triggered when /s/ is appended to a sibilant-final word:

(3) stretchs ↦ stretches

The affixation map and the *e*-insertion map have a string in common: the output of the first is the input to the second. This allows the two maps to be combined by *composition*, a means of cutting out the intermediate step and representing the entire map with a single FST:

(4) a. Before composition: stretch+V+3P+Sg ↦ stretchs, stretchs ↦ stretches
b. After composition: stretch+V+3P+Sg ↦ stretches

By extension, any series of ordered FSTs can be composed into a single FST that maps tagged lexical items to their surface pronounced forms. This is possible because the regular relations have the property of being closed under composition.[7]

In addition, the *inverse* of this same FST (i.e., the FST in which the input and output of each transition is flipped) can be used to decompose a surface form into its component morphemes/tags:

[5] For a more comprehensive introduction to the finite state formalism and its application to phonology and morphology, readers are directed to Beesley and Karttunen (2003) and Roark and Sproat (2007).

[6] Note that the strings in this example are orthographic instead of phonemic, which is the norm for a system designed to analyze text.

[7] Formally, this means that if R_1 and R_2 are regular relations and $(x, y) \in R_1$ and $(y, z) \in R_2$, then there exists another regular relation R_3 such that $(x, z) \in R_3$.

(5) stretches ↦ stretch+V+3P+Sg, stretch+N+Pl

Such systems have wide application in various areas of natural language processing. The fact that they can be implemented with FSTs is due to the theoretical foundations of the results presented in this paper. First, as will be explained in more detail in Sect. 3.3 below, phonological rules can be modeled with regular relations. Second, regular relations are closed under composition, meaning the composition operation that combines the component FSTs is guaranteed to produce a well-defined and correct FST for the complete map. Thus the classification of phonological processes and morphological operations like affixation as regular has the practical advantage that morpho-phonological systems can be efficiently implemented as finite state.

The interests of the current paper lie more in pushing the boundaries of these influential previous findings. Many unattested and implausible maps are also regular (and therefore can also be implemented as finite state). The theoretical question of interest is then how far 'below' regular can we go while still accommodating the range of attested maps. The results presented below indicate that both phonological and morphological maps belong to subregular classes of functions, which not only provide a better fit to the observed typology but also (as mentioned above) enable efficient learning results.

Before turning to the analyses, however, the next section will briefly review the previous theoretical results on the computational nature of syntactic (Sect. 3.1), morphological (Sect. 3.2), and phonological (Sect. 3.3) patterns.

3 Computational analyses of natural language patterns

3.1 Syntax

The distinction between regular and non-regular was first applied to natural language patterns by Chomsky (1956), who situated several English syntactic patterns on the hierarchy of complexity classes shown in (6).

(6) Chomsky Hierarchy
finite ⊂ regular ⊂ context-free ⊂ context-sensitive ⊂ recursively enumerable

In particular, Chomsky identified English syntactic patterns with the same type of dependency needed to recognize the non-regular $a^n b^n$ language mentioned above. As an example, consider the sentence frame 'If S_1, then S_2', where S_1 and S_2 are sentences of English. A dependency exists between 'if' and 'then' in that a sentence that begins with 'if' must at some point also contain a 'then'. If S_1 is itself a sentence of the form 'If S_3, then S_4' (i.e., 'If [If S_3, then S_4], then S_2'), we have two 'if's' that must be followed at some point by two 'then's'. And so on, such that to determine whether the sentence is well-formed requires keeping track of the same kind of information needed to determine whether 'aaabbb' is in the language $\mathbb{L}_n$. And for the same reasons, the 'if… then' structure describes a non-regular language. This was taken as evidence that English itself is a non-regular language.

This argument was later recognized as fallacious, since a regular language can contain a context-free language as a subset (see Daly 1974; Mohri and Sproat 2006).

But additional evidence that syntax is non-regular came from Shieber (1985) and Kobele (2006). Shieber's argument comes from case-marking dependencies in Swiss German's cross-serial construction:

(7) Swiss German (Shieber 1985)
Jan säit das mer em Hans es huus hälfed aastriiche
Jan says that we Hans-DAT the house-ACC helped paint
'Jan says that we helped Hans paint the house.'

In (7), two verbs and their respective objects appear in the order 'object$_1$ object$_2$ verb$_1$ verb$_2$'. The semantic dependencies are encoded syntactically with case-marking, as verb$_1$ (*hälfed*) marks object$_1$ (*Hans*) with dative case and verb$_2$ (*aastriiche*) marks object$_2$ (*huus*) with accusative case. More abstractly, this means that a dative-marked NP (let's call it a) must precede an accusative-marked NP (let's call it b), which in turn must precede a dative-marking verb (c), which in turn must precede an accusative-marking verb (d). More generally, these precedence relations still hold if the sentence contains additional NPs and verbs of these categories, such that the sentences of the language can be represented as $a^m b^n c^m d^n$, where m and n are integers. Crucially, the number of a's must match the number of c's and likewise the number of b's must match the number of d's. Such a language is known to be non-context-free (i.e., cannot be generated by a context-free grammar) (Hopcroft et al. 2000).

Likewise, Kobele (2006) discusses serial verb constructions in Yoruba relativized predicates:

(8) Yoruba (Kobele 2006)
a. Rira ti Jimọ ọ ra adiẹ
buying TI Jimọ HTS buy chicken
'the fact/way that Jimọ bought a chicken'
b. Rira adiẹ se ti Jimọ ọ ra adiẹ se
buying chicken cook TI Jimọ HTS buy chicken cook
'the fact/way that Jimọ bought the chicken to cook'

These constructions are analyzed as involving copying, and Kobele (2006) argues that the fact that relative predicates can themselves contain relative predicates means the copying is iterative (i.e., copying of copies can occur). Furthermore, there is no principled upper bound on the number of relative clause embeddings, and therefore by extension on the amount of material copied. The exact same mechanism Kobele (2006) proposes to account for this could generate the context-sensitive language ww^r (i.e., all strings are anagrams).

Thus there is evidence that—at least when it comes to syntax—natural language patterns are not only non-regular, but can be as complex as context-sensitive. But what about non-syntactic patterns? Analyzing patterns computationally in different domains is one avenue to understanding more about how these domains differ. The next subsection reviews the results when similar analyses were applied to morphological patterns.

3.2 Morphology

Focusing on well-formed words instead of sentences, Langendoen (1981) hypothesized that no language's word formation component requires more power than regular. But Culy (1985) provided evidence from two constructions in Bambara to suggest otherwise: the Noun *o* Noun construction (shown in (9)), which is only acceptable with two identical nouns, and an agentive construction (Noun+Verb+*la*, as in (10a)) that can be used recursively (as in (10b)).

(9) Bambara (Culy 1985)

a. wulu 'dog'
b. wulu o wulu 'whichever dog'

(10) a. wulu+nyini+la 'dog searcher'
b. wulunyinila+nyini+la 'one who searches for dog searchers'

In addition, nouns derived via the agentive construction can themselves be used in the Noun o Noun construction:

(11) wulunyinila o wulunyinila 'whichever dog searcher'

Culy shows that the potential for recursion in the agentive construction and the requirement that the nouns be identical in the Noun *o* Noun construction make the pattern equivalent to $a^m b^n a^m b^n$, which is more powerful than context-free (and by extension more powerful than regular).[8]

Gazdar and Pullum (1985) point out that all known cases of non-regular word formation (like Culy's Bambara example) involve reduplication. Though the presence of reduplication in a language means the entire word formation component is non-context-free (as shown by Culy 1985), they note that recognizing whether the first part of the string is equal to the second can be achieved with the same kind of parsing algorithms that are used for context-free languages (e.g., CKY). This suggests that the right characterization of word formation is 'regular with reduplication', though this class lacks a formal characterization.

Carden (1983) argues against the focus on weak generative capacity, as the output of the word formation component must include the relevant structure (i.e., trees/bracketings) to be of use to the phonological and semantic components. When instead focusing on this strong generative capacity (complexity of the structures assigned to strings), he argues morphology is indeed more powerful than regular. He points to Bar-Hillel and Shamir (1960)'s examples of the recursive nature of English shown below:

(12) a. missile
b. anti-missile missile
c. anti-anti-missile missile missile
d. etc.

[8] A reasonable follow-up question would be whether these patterns are in fact syntactic, where we expect to find non-regular phenomena. Culy gives evidence based on the tone pattern of these nouns that suggests it is in fact a morphological phenomenon.

Based on the weak generative capacity of this construction—the non-regular set of strings $\{(\text{anti})^n\text{missile (missile)}^n,\ n \geq 1\}$—Bar-Hillel and Shamir (1960) argue that English morphology is not limited to regular.[9] Carden (1983) concurs with this conclusion, but for a different reason, arguing that assigning the correct structure to these forms requires center-embedding. Center-embedding in turn requires both left- and right-branching structures, whereas regular grammars can do either left- or right-branching, but not both.

This distinction between weak and strong generative capacity has a significant impact on our understanding of the computational nature of syntactic and morphological patterns. The research in computational phonology reviewed in the next section introduced another important distinction: classifying sets of strings versus the maps (i.e., relations or functions) that actually generated those strings.

3.3 Phonology

The foundational results of Johnson (1972), Koskenniemi (1983), and Kaplan and Kay (1994) showed that phonological rules of the form $A \rightarrow B \ / \ C \_\_ D$ are *regular relations*, provided they do not re-apply to the locus of the structural change. The regular relations parallel the regular languages: while a regular language is a set of strings, a regular relation is a set of string pairs (i.e., $\{(w_1, w_2), (w_3, w_4), \ldots\}$). The first member of the string pair is *related to* the second member in some well-defined way. For example, the string pair $(abcd, ad)$ is a member of the relation in which the first string is mapped to a string that contains only its first and last characters. When it comes to a phonological regular relation, the string pair is often an underlying representation and the surface representation it is mapped to by a particular generalization/process/rule (e.g., (UR, SR)). In automata-theoretic terms, the regular relations are those relations describable with FSTs. As noted above, because the regular relations are closed under composition, a single relation can in fact describe the direct UR-SR map of an entire set of ordered rewrite rules (see Kaplan and Kay 1994).

The result that phonological rules are regular relations was significant for at least two reasons. One, it indicated that phonology is less computationally complex than syntax (see Bromberger and Halle 1989; Heinz and Idsardi 2011, 2013). Two, it revealed that the context-sensitive Sound Pattern of English (SPE)-style (Chomsky and Halle 1968) rewrite rules being used at the time to describe phonological grammars were more computationally expressive than necessary. To the extent that a theory of phonology should predict the set of patterns that are actually possible, the computational analysis of phonological rules revealed that a significant property of phonology was being missed.

This result for phonological rules might lead one to assume that phonotactic patterns are also regular, given that the language that results from a regular relation (in this case the language of surface forms) is itself regular. However, there is strong evidence that in fact phonotactic patterns are best characterized as *subregular* languages

[9] See Langendoen (1981) for an argument against their conclusion.

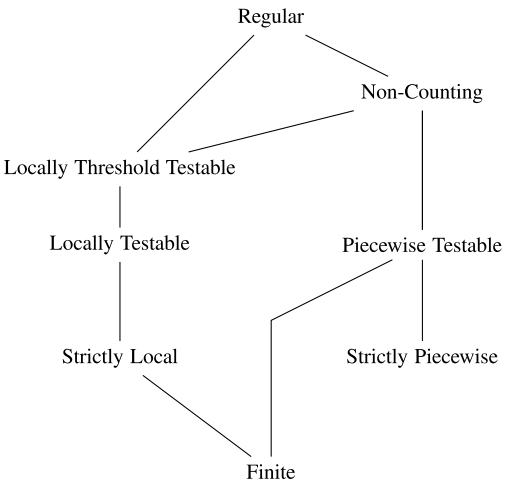


Fig. 4 Subregular hierarchy of formal languages (Rogers and Pullum 2011)

(Heinz 2007, 2009, 2010; Heinz et al. 2011). In other words, if regular was the upper bound on the computational complexity of phonotactics, then a greater variety of patterns should be attested cross-linguistically.

To pursue the hypothesis that phonotactics are actually subregular required a more articulated hierarchy than the one in (6), one with options between Regular and Finite.[10] The subregular hierarchy of languages, shown in Fig. 4 (McNaughton and Papert 1971; Rogers and Pullum 2011; Rogers et al. 2013), provided exactly that. Unlike (6), Fig. 4 includes several regions between Regular and Finite. Both the horizontal and vertical orientations of these regions are meaningful. Vertical lines connecting regions indicate that one region properly includes the other (e.g., all Locally Testable languages are also Locally Threshold Testable, etc.). The two branches originating at Regular are distinguished by the interpretation of the constraints that define the language (i.e., immediate successor versus general precedence).

A comparison between the Strictly Local (hereafter SL) and Strictly Piecewise (SP) languages will clarify this distinction between successor and precedence. Consider a language with the alphabet $\Sigma = \{\text{T, D, V, N}\}$ that prohibits the segment D from being the last segment of a string. This infinite language can be represented with a finite *grammar*, $\mathbb{G}_{SL}$, that lists the *forbidden substrings*. In this example, $\mathbb{G}_{SL} = \{\text{D}\ltimes\}$ (following Rogers and Pullum (2011), let $\rtimes$ and $\ltimes$ represent the start and end of word boundaries, respectively). The language is the set of strings that do not contain any of the substrings in $\mathbb{G}_{SL}$. Languages that can be defined in this way (i.e., with a grammar of contiguous substrings of bounded length) belong to the SL region. A given SL language is actually k-SL, where k is the length of the longest substring in the grammar. This example is then 2-SL.

[10] A finite language is simply a finite set of strings. The grammar for such a language would not have an infinite generative capacity. For this reason finite formal languages (and by extension finite relations) have little to no theoretical interest for natural languages, under the assumption that there is no upper bound on the length of words in a human language (i.e., human languages are infinite).

The difference between SL and SP is that in the latter the grammar includes *subsequences* instead of *substrings*. A subsequence of a string can be non-contiguous; for example, T…V is a subsequence of the string DTDVD. Subsequences track the precedence relations of the symbols in a string. As an example, consider a language for which $\Sigma = \{s, V, ʃ\}$, such that the symbol 's' can never precede the symbol 'ʃ' in a string. In other words, the valid strings of this language are those that do *not* contain the subsequence s…ʃ. Such a language is 2-SP, since the forbidden subsequence is of length 2.

These examples of SL and SP languages correspond intuitively to local and long-distance phonotactic constraints, respectively (Heinz 2010). The SL example is a language that enforces final devoicing (when we interpret the alphabet as D = voiced obstruent, T = voiceless obstruent, V = vowel and N = nasal). And the SP example is of course a language that enforces sibilant harmony, such as Navajo:[11]

(13) Navajo (Sapir and Hoijer 1967; Hansson 2001; Heinz 2010)

a. sì-tí 'he is lying'
b. ʃì-ɣìʃ 'it is bent, curved'
c. *sì-ɣìʃ

Thus when it comes to phonotactics there is evidence that the observed patterns are not only subregular, but fall into the most restrictive regions of the hierarchy in Fig. 4: Local phonotactics are SL and long-distance phonotactics are SP. However, long-distance patterns with blocking are exceptions to this generalization (Heinz 2010). Such patterns still do not require the full power of the Regular class, provided they can be defined over a *tier* (i.e., a subset of Σ that includes only those segments participating in the phonotactic constraint). In that case the blocking pattern can be described as a Tier-based Strictly Local (TSL) language (Heinz et al. 2011; McMullin 2016). As the name implies, a TSL language is defined with SL-type constraints over only those segments on the tier (i.e., all other segments are ignored). The TSL region is not represented in Fig. 4, but it is properly contained by Regular (Heinz et al. 2011).

Investigations of the computational nature of phonology have also examined phonological UR-SR maps to determine the extent to which they too are subregular. Since the subregular hierarchy in Fig. 4 is a hierarchy of formal languages, it cannot be used directly to study maps, which again are not sets of strings, but sets of string pairs. For example, the final devoicing case above was described as the set of strings without voiced obstruents in word-final position. The corresponding map would be one that, given a string that *does* contain a voiced obstruent in word-final position, maps that string to one with the respective voiceless obstruent in word-final position. So the final devoicing map is $D = \{$(TV, TV), (DV, DV), (DVN, DVN), (DVD, DVT), … $\}$.[12] The subregular hierarchy of maps, shown in Fig. 5 (Mohri 1997; Chandlee 2014; Chandlee et al. 2015), is not as fully developed as the one for languages, but it has lead to several key results for phonology.

[11]This is a simplification of the Navajo facts. More generally, [+anterior] sibilants cannot precede [−anterior] sibilants, and vice versa.

[12]Note that strings that do not contain a voiced obstruent in word-final position are simply mapped to themselves. In other words, D is a *total* function defined for all strings from Σ^*, not just those that satisfy the structural description for final devoicing.

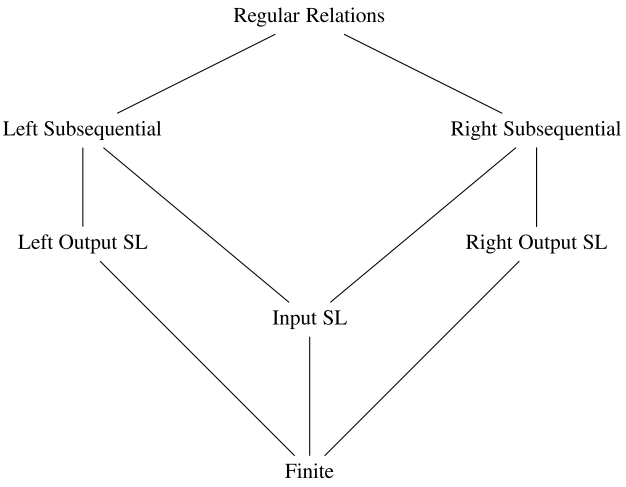


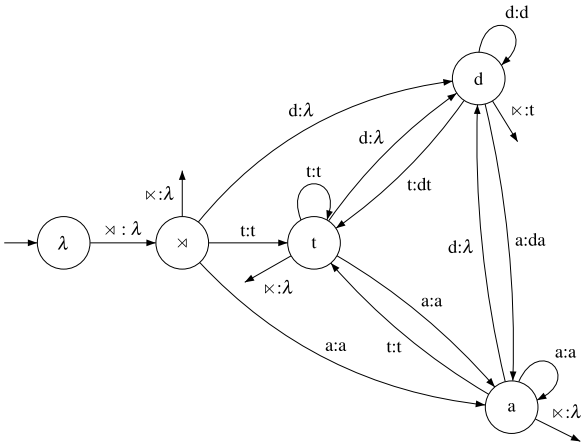
Fig. 5 Subregular hierarchy of maps

Comparing Figs. 4 and 5, we see both align with Regular at the top and Finite at the bottom. Note that the SL languages have three counterparts in the maps hierarchy: left Output Strictly Local (LOSL), right Output Strictly Local (ROSL), and Input Strictly Local (ISL). These will be discussed in more detail below. The left and right subsequential functions are not known to correspond to any region of the hierarchy of languages. The map counterparts to the other regions of the language hierarchy (i.e., Non-counting, Locally Threshold Testable, Locally Testable, Piecewise Testable, and Strictly Piecewise) remain to be discovered.

Chandlee (2014), Chandlee et al. (2015), and Chandlee and Heinz (2018) show that phonological maps that correspond to local processes (i.e., processes for which the target and triggering context form a contiguous substring of bounded length) can all be classified in one of the SL regions of the maps hierarchy. In the finite state formalism, this means these types of phonological generalizations can be described with FSTs that have the characteristic properties of the LOSL, ROSL, and/or ISL classes. We will continue with final devoicing as an example to demonstrate what this means.

Final devoicing is a straightforward case of a 'local' process, in that the trigger of the process (word-final boundary) and the target (voiced obstruent) form a contiguous substring of bounded length (i.e., 2). In other words, whether or not the process applies can be determined solely by examining whether an input string contains the substring D⋉. This 'bounded' nature of the map, plus the fact that the needed information is present in the input, means we can model it as a 2-ISL function. Again, this in turn means it can be represented with a 2-ISL FST, which is shown in Fig. 6.

A few notes on the FSTs that will be presented in the remainder of the paper. The start state is always λ, which represents the 'empty string' of zero symbols. Starting in this state means no symbols have been read. All input strings are assumed to be augmented with ⋊ and ⋉, which are not included in Σ. Thus an input string $w \in \Sigma^*$ is treated as $\rtimes w \ltimes$ by the FST. The start state always has a single outgoing transition



Input:		⋊		d		a		t		a		d		⋉	
States:	λ	→	⋊	→	d	→	a	→	t	→	a	→	d	→	
Output:		λ		λ		da		t		a		λ		t	

Fig. 6 2-ISL FST for final devoicing

Fig. 7 Path through FST in Fig. 6 for input *datad*

on the input symbol ⋊—in most cases the output of this transition is λ, though we will see cases in the survey of morphological maps in which it plays a more important role.[13] In addition, each state (except for λ) has an extra transition with ⋉ as the input symbol. These *final output transitions* are only taken when the end of the input string is reached; their output is then appended to the end of the output string.

For simplicity, the FST in Fig. 6 assumes the following segment inventory: $\Sigma = \{\text{d, t, a}\}$.[14] For the input string *datad*, this FST follows the path shown in Fig. 7. As mentioned above, the fact that final devoicing can be modeled with any FST is sufficient to classify it as a regular relation. Its further classification as ISL (and therefore subregular) depends on a few special properties of the FST in Fig. 6. First, it is *deterministic*, which means each state has at most one outgoing transition for each possible input symbol. FSTs in general can be *non-deterministic*, meaning a state could have multiple transitions for a given alphabet symbol. Some regular relations can only be modeled with non-deterministic FSTs, but all ISL functions can be modeled deterministically. Second, the ISL FST includes states for each possible input sequence of length $k-1$ (in this example again $k=2$, so there are states for each sequence of length 1). FSTs in general can have states that represent other types of information,

[13] Note that since ⋊ is not part of Σ and is therefore guaranteed to only appear once at the start of the string, the λ state and the ⋊ state could also be collapsed with the ⋊ transition being a self-loop. Keeping the two states distinct is motivated by greater transparency in how they represent the pattern in question. See also Chandlee et al. (2015) for reasons why, at least in OSL FSTs, a distinct ⋊ state is necessary.

[14] The fact that the map is ISL does not depend on this reduced alphabet. It would still be ISL, for the same value of k, if the alphabet included the complete segment inventory for a particular language. The FST in that case would just have more states and therefore be less readable.

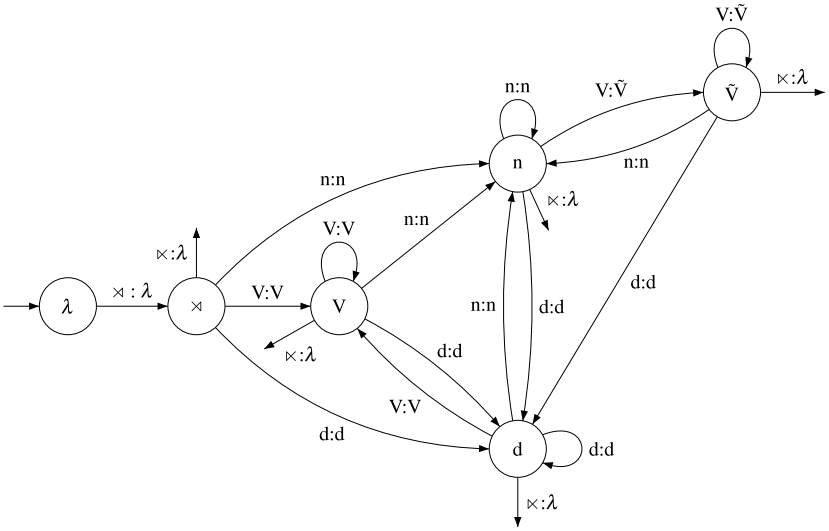


Fig. 8 2-LOSL FST for progressive nasal spreading

but ISL FSTs can only keep track of the most recently read $k-1$ symbols. No other information can be used to determine what to output at any given time. This is the essentially 'local' nature of an ISL map.

The Output SL maps are very similar to the ISL ones, except that the FST tracks the recent output instead of the input. This is needed to model processes in which the trigger is present in the output and not the input. An example is nasal spreading, like in Johore Malay (Onn 1980).

(14) Johore Malay
f(pəŋawasan) = pəŋãwãsan 'supervision'

Under the assumption that the nasalization proceeds iteratively, such that the first *a* is nasalized because of the preceding nasal, and then it in turn nasalizes the following glide, etc., the triggers for the nasalization of the glide and the second *a* are only present in the output, not the input. Therefore an ISL FST can't model this process, since, again, it can only pay attention to the recent input. An OSL FST can, however, model this process. Since the process is progressive (i.e., proceeds left-to-right), it is modeled with a left OSL FST, which reads the input from the left to the right. This FST is shown in Fig. 8. Regressive iterative processes are likewise classified as right OSL; a right OSL FST reads the input string from the right to the left.[15]

Again for readability, the FST in Fig. 8 is defined for the reduced alphabet of {n, d, V}, where V is any [+vocalic] segment. Note that once in state *n*, if a V is read the output is nasalized Ṽ, and that transition crucially leads to a state Ṽ. Were this an ISL FST, that transition would go to state V. In that case, nasalizing any additional

[15]ISL FSTs are not designated as left or right because when paying attention to the input the same map will result regardless of whether the string is read from the left or the right. For more on this distinction, see Kaplan and Kay (1994), Hulden (2009a), Heinz and Lai (2013).

V's would require first seeing another *n*. But since the OSL FST follows the output, additional V's can be nasalized directly from state $\tilde{V}$.

Maps in which an unbounded number of segments intervenes between the target and trigger are neither ISL nor OSL. An example of unbounded consonant agreement in Kikongo is shown in (15).

(15) Kikongo (Meinhof 1932; Odden 1994; Rose and Walker 2004)

a. f(tu**n**iki**d**i) = tu**n**iki**n**i 'we ground'

b. f(kudu**m**ukisi**l**a) = kudu**m**ukisi**n**a 'to cause to jump for'

While unbounded maps like long-distance consonant agreement, long-distance consonant dissimilation, and vowel harmony are not ISL/OSL, they are subsequential and therefore still subregular (Payne 2017; Gainor et al. 2012; Heinz and Lai 2013).

In sum, previous investigations into the computational nature of phonological maps have provided a set of categories for classifying patterns as well as substantial evidence that phonological maps are subregular. The next section applies these same categories toward a comparable investigation of morphological maps.

4 Computational analyses of morphological patterns

This section extends the computational analyses of phonological maps presented above to morphological maps, which are functions that take an input string and produce an output string by applying some type of morphological operation. First various types of concatenative morphology are discussed, including (non-reduplicative) affixation, partial reduplication (both 'local' and 'non-local' varieties), and total reduplication. Then a couple of types of non-concatenative morphology are discussed, including featural affixation, in which no segments are added to the word but rather a floating feature is expressed on the existing segments of the word, and truncation. Templatic morphology and compounding are not given a full analysis here, but some comments will be offered in Sect. 5 on how these operations differ from those presented in this section.

4.1 Affixation

We begin with the straightforward case of English ɪŋ-suffixation, by which the string *ɪŋ* is attached to the end of a verb to encode the present progressive tense. An example of this map is shown in (16) (repeated from (1)).

(16) f_{prog}(spik) = spik+ɪŋ

Before proceeding to the classification of this map, two important assumptions of these analyses are stated. First, the output of the map encodes the morpheme boundaries (here with the symbol '+'), under the assumption that such information crucially defines the context for at least some phonological maps. Second, the maps are considered to be *total functions*, meaning they treat all input strings the same, regardless of whether the string corresponds to an actual input of an actual speaker. In other

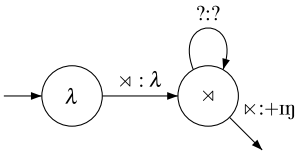


Fig. 9 1-ISL FST for English ŋ-suffixation

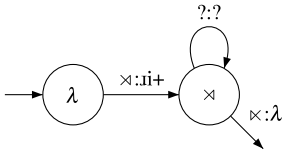


Fig. 10 1-ISL FST for English ɹi-prefixation

words, out of the possible inputs $W = \{$ɹʌn, dɪŋk, kɔfi, ɑɑɑɑ$\}$, only the first two are 'valid' inputs to ŋ-suffixation in the sense that they are strings of English phonemes that correspond to verbs. However, the suffixation map does not make this distinction and will apply equally to all four strings: $f_{+ŋ}(W) = \{$ɹʌn+ŋ, dɪŋk+ŋ, kɔfi+ŋ, ɑɑɑɑ+ŋ$\}$. This allows us to analyze the computational properties of the map itself independently of how it is actually used within the larger system. To put it a different way: the analysis of computational complexity is focused on the morphological operation itself (i.e., appending a string), not the determination of whether the input actually corresponds to a verb of English.

The map exemplified in (16) is ISL for $k = 1$, and its FST is shown in Fig. 9. Following Beesley and Karttunen (2003), the '?' transition encompasses all segments not represented on other transitions (in this case that is all segments in Σ). The ?:? self-loop on state ⋊ then effectively outputs the entire input string unchanged, up until it reads the end of word marker ⋉, at which point the suffix is appended.

Prefixation is also 1-ISL. Consider the example of the English prefix *re-*, which again attaches to verbs:

(17) a. f_{re}(ɹid) = ɹi+ɹid
b. f_{re}(wɑtʃ) = ɹi+wɑtʃ

A 1-ISL FST for this function is shown in Fig. 10. The prefixation takes place on the output side of the ⋊ transition, after which all additional input is outputted unchanged by a ?:? transition.

Summarizing these two examples, we see that both suffixation and prefixation are 1-ISL maps: the former is achieved via the final output function (i.e., the transition on ⋉) and the latter is achieved with the first transition on ⋊. If we combine these two options in a single FST, we can model circumfixation. An example comes from Chickasaw (Fromkin et al. 2014), in which negation is achieved by prefixing *ik-* and suffixing *-o*:

(18) Chickasaw
a. f_{neg}(chokma) = ik+chokm+o (He is good. → He isn't good.)
b. f_{neg}(lakna) = ik+lakn+o (It is yellow. → It isn't yellow.)

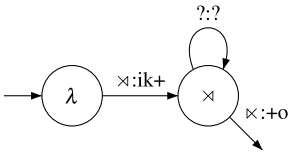


Fig. 11 1-ISL FST for Chickasaw circumfixation

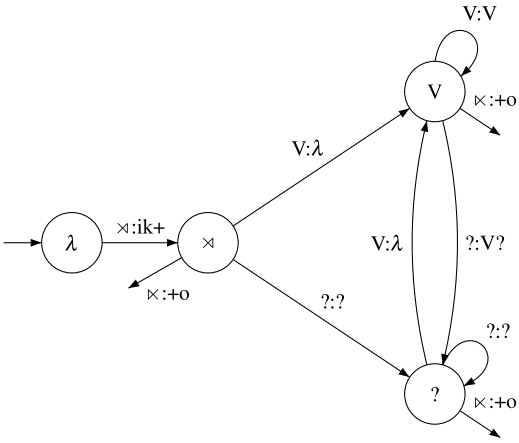


Fig. 12 2-ISL FST for Chickasaw circumfixation and vowel deletion

Putting aside for a moment the deletion that resolves vowel hiatus (i.e., $ao \mapsto o$), this circumfixation map can be modeled with the 1-ISL FST in Fig. 11. Comparing this FST with those in Figs. 9 and 10, we see that for circumfixation the transitions on both $\rtimes$ and $\ltimes$ contribute non-empty strings to the output.

Again, as the current goal is to classify various categories of morphological maps in terms of their computational properties, this example suffices to demonstrate that circumfixation *in isolation of the phonology* is 1-ISL. But to cover the Chickasaw data in full, we now briefly demonstrate how the classification is affected by the vowel deletion triggered by the circumfixation. As discussed in the previous section, deletion maps with local triggers (such as deletion to resolve vowel hiatus, as in the case at hand), are ISL maps. Thus here we have a dataset that reflects two ISL maps, one for the circumfixation and one for the vowel deletion. Though all of our examples of ISL maps so far have dealt with a single process or operation, in fact a single ISL map can describe multiple processes/operations. The ISL FST in Fig. 12, for example, models both circumfixation and vowel deletion. Notice that the k-value has increased to 2.

The FST in Fig. 12 achieves the prefixation of *ik-* in the same way as in Fig. 11. After that, it moves to the appropriate state depending on the first segment of the word: state V if that segment is a vowel and state ? otherwise. The output for the V transition is λ, indicating that the vowel is deleted under the assumption that the suffix *-o* will be appended. If that is not the case, meaning another non-vowel symbol follows the vowel, then the vowel is 'returned' on the subsequent ? transition to state ? (along with ? itself). The FST proceeds in this way based on all additional V and ? segments, until it does reach the end, at which point the suffix is appended via the $\ltimes$ transition just as before. A example path for input *lakna* is given in Fig. 13. Thus

Input:		⋊		l		a		k		n		a		⋉
States:	λ	→	⋊	→	?	→	V	→	?	→	?	→	V	→
Output:		ik+		l		λ		ak		n		λ		+o

Fig. 13 Path through FST in Fig. 12 for input *lakna*

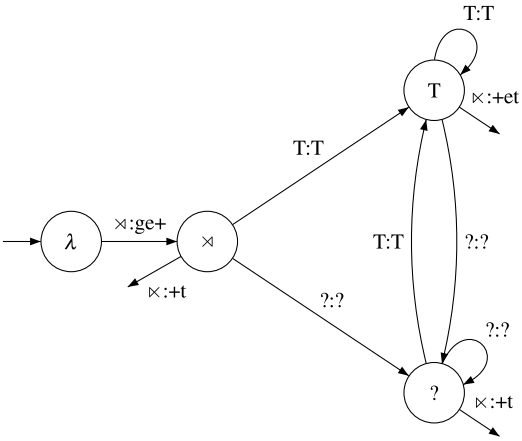


Fig. 14 Fragment of ISL FST for German circumfixation

in this case the interaction of circumfixation and vowel deletion does not change the computational classification (i.e., it is still an ISL map).

Another example of circumfixation that also involves some allomorphy is the German past participle, shown in (19). This map prefixes *ge-* and suffixes *-t*, unless the stem ends in (1) an alveolar stop or (2) a nasal that is preceded by a non-liquid consonant, in which case the suffix is *-et*. For readability, the portions of the FST responsible for these two generalizations will be shown separately (the complete FST for the entire map, for which $k = 3$, is included in an Appendix).

(19) German

a. f_{past}(mach) = ge+mach+t ('make' → 'made')
b. f_{past}(koch) = ge+koch+t ('cook' → 'cooked')
c. f_{past}(miet) = ge+miet+et ('rent' → 'rented')

The first generalization—that the suffix *-t* is *-et* when the stem ends in an alveolar stop—is modeled with the FST fragment in Fig. 14. In this FST the symbol T = {t, d} and ? again abbreviates everything else. Note that the allomorphy is handled straightforwardly with the final output function: in state T the appended suffix is *-et*.

The second generalization—that the *-et* allomorph is also used when the stem ends in a consonant cluster of a non-liquid followed by a nasal—is described with the portion of the FST in Fig. 15. In this FST the symbol N is used for any nasal and L is used for any liquid. Again '?' represents all other segments. The *-et* allomorph is appended at state '?N', which corresponds to any stem-final sequence of a non-liquid and a nasal. Stems that end in a liquid-nasal cluster will end in state LN, were the *-t* allomorph is appended instead. Thus the distribution of the two suffixes can be

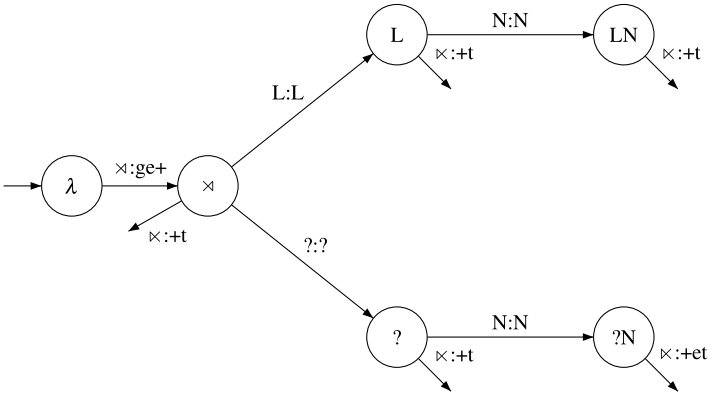


Fig. 15 Fragment of ISL FST for German circumfixation

achieved by keeping track of the last two segments of the input string, making it a 3-ISL map.

This leaves infixation. McCarthy and Prince (1993, 1996) identify two types of infixation within their framework of prosodic circumscription: negative and positive circumscription. In negative circumscription, a prosodic constituent is skipped over or put aside while a morphological operation applies to the remainder of the string. An example is *um*-infixation in Tagalog, shown in (20) (French 1988; McCarthy and Prince 1993; Orgun and Sprouse 1999). The infinitive affix *um* appears as a prefix before vowel-initial stems and after the initial onset of consonant-initial stems.

(20) Tagalog

a. f_{inf}(abot) = um+abot ('reach for' → 'to reach for')
b. f_{inf}(sulat) = s+um+ulat ('write' → 'to write')
c. f_{inf}(gradwet) = gr+um+adwet ('graduate' → 'to graduate')

Because the infixation map can determine the correct placement of the infix by examining at most the first three segments of the string, it is a 4-ISL map (the fourth symbol is ⋊). The FST in Fig. 16 models Tagalog *um*-infixation. Since once the infix is placed the rest of the string is just outputted unchanged, all subsequent states have been collapsed to a single '?' state for readability.

In contrast, in positive circumscription the infix attaches to a prosodic constituent. For example, in Ulwa (Bromberger and Halle 1988; Hale and Blanco 1989; Sproat 1992; McCarthy and Prince 1993; Roark and Sproat 2007) the possessive is formed by infixing a pronoun after the first syllable if it is heavy, otherwise after the second syllable.

(21) Ulwa

a. f_{pos}(bas) = bas+ka ('hair' ↦ 'his hair')
b. f_{pos}(ki:) = ki:+ka ('stone' ↦ 'his stone')
c. f_{pos}(sana) = sana+ka ('deer' ↦ 'his deer')

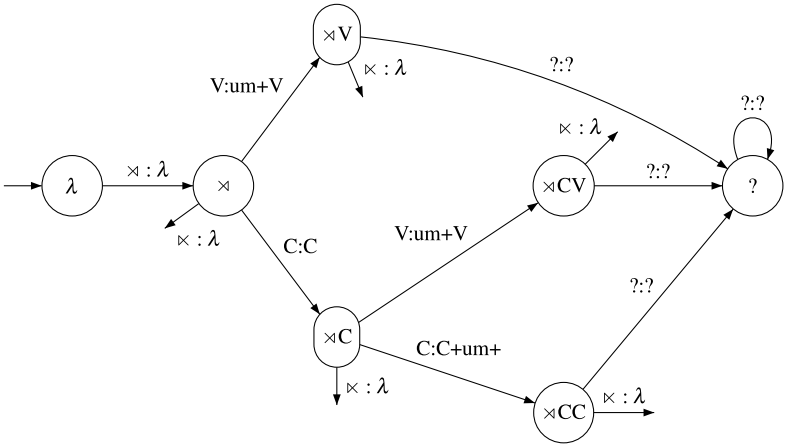


Fig. 16 4-ISL FST for Tagalog *um*-infixation

More generally, the possessive affix is attached after the first iambic foot. The options for the form of this foot are disyllables in which the first vowel is short and monosyllables with either a long vowel or a coda. These options are shown in template form in (22):

(22) a. (C)VCV(V)
b. (C)VV
c. (C)VC

This operation requires examining at most the first 6 segments of the string (the first segment being ⋊).[16] If the first vowel is followed by another vowel, then we know we have a heavy monosyllabic foot (22b) and so the infix can be attached at this point (⋊CVV ↦ ⋊CVV+ka). Otherwise, we have to keep going to determine whether we have a closed monosyllabic foot (22c) or a disyllabic foot (22a). After CVC, if what follows is another C then the infix can be attached between the two consonants: ⋊CVCC ↦ ⋊CVC+ka+C. If what follows is a V, then one additional segment after that must also be examined; if it's the second half of a long vowel, the infix is attached after the long vowel (⋊CVCV$_1$V$_1$ ↦ ⋊CVCV$_1$V$_1$+ka), but if it's a different vowel or a consonant, the infix is attached prior to it (⋊CVCV$_1$V$_2$ ↦ ⋊CVCV$_1$+ka+V$_2$, ⋊CVCVC ↦ ⋊CVCV+ka+C). These last two cases represent the upper bound on the number of segments that must be read, 6, to model the operation as ISL.

One might object to this analysis of Ulwa because it just examines the segments directly and does not actually make use of the metrical structure. But again the goal here is to simply answer the question of whether or not the map can be modeled with the restrictions of an ISL function, and the answer to that question in this case is yes. There do, however, exist cases of affixation conditioned by metrical and/or prosodic structure that cannot be modeled as ISL based on segments alone (or even at

[16]This is assuming long vowels are represented as VV; if the alphabet instead includes a V: symbol then only the first 5 segments need to be examined.

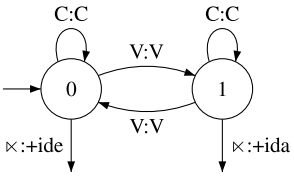


Fig. 17 Left subsequential FST for Sami Illative Plural

all). Three cases will be reviewed here, from Sami, Yidiɲ, and Tagalog, all of which require some mechanism for counting the number of syllables in the word.[17]

In Sami, the illative plural has two allomorphs that are selected based on whether the noun has an even or odd number of syllables.

(23) Sami Illative Plural (Bergsland 1976; Hargus 1993)

a. f_{ilpl}(čiega) = čiega+ide 'corner'
b. f_{ilpl}(mállási) = mállási+ida 'feed'

This is suffixation, which was analyzed above as being 1-ISL. But tracking whether a string has an even or odd number of syllables is beyond the ability of any ISL FST. It is however, subsequential and therefore still subregular. Figure 17 presents a left subsequential FST for the Sami Illative Plural operation. This 2-state FST keeps track of the even/odd parity of the number of syllables by counting the vowels modulo 2 (V abbreviates the set of vowels and diphthongs). The FST will always be in state 0 when the vowel count is a multiple of 2; otherwise it will be in state 1. The final output function from each state appends the appropriate suffix.

Similarly, in Yidiɲ, a final syllable deletion process targets words with an odd number of syllables, with the added restriction that the post-deletion form must end in one of {l, r, ɽ, y, m, n, ɲ, ŋ}. Examples are given below; the vowel lengthening in (24a) is due to a penultimate lengthening process that also targets words with an odd number of syllables.[18]

(24) Yidiɲ Final Syllable Deletion (Dixon 1977; Hayes 1999)

a. buɲa+ŋgu ↦ buɲaːŋ 'woman (ergative)'
b. gindanu+ŋgu ↦ gindanuŋgu 'moon (ergative)'

The result is two allomorphs that are selected based on the even/odd parity of the number of syllables in the stem. For the ergative these are *-ŋ* and *-ŋgu*, though the same pattern is observed with a number of other suffixes. Though the suffixation operation itself is still a simple 1-ISL function, the subsequent phonological changes of vowel lengthening and syllable deletion still need to be addressed, as their conditioning on syllable number reflects global information about the string that falls beyond the capability of ISL.

As with the Sami data above, the map that achieves the suffixation as well as the lengthening and deletion processes can be modeled with a left subsequential FST, which has the ability to determine whether an input string contains an even or odd

[17] Thanks to an anonymous reviewer for bringing these cases to my attention.

[18] See Hayes (1982, 1999) for additional examples of processes in this language that depend on the number of syllables in the word.

number of syllables. The analysis of Hayes (1982), however, provides an alternative approach in which the map is still ISL. He argues that the environment for lengthening and deletion is assessed by whether or not the word ends with an unparsed syllable. For example, lengthening applies in (24a) but not (24b) because the former has a final unparsed syllable (which in turn means the whole word has an odd number of syllables):

(25) a. [buŋaŋ]gu ↦ [buŋaːŋ]gu
b. [ginda][nuŋgu] ↦ [ginda][nuŋgu]

Such a map is ISL, *provided the input is already parsed for foot structure*. Because the substring of interest includes both segmental material and foot structure (here represented with bracketing), the k-value of this map is 7:

(26) V(C)]CV(C)⋉ ↦ V:(C)]CV(C)

The consequences of allowing such non-segmental markup in the input (and alternatively, the extension of this framework to non-linear representations), raises important questions that are being left for future work.

Lastly, in addition to the *um*-infixation case analyzed above, Tagalog also has perfective *in*-infixation that follows the two patterns described in (27) (examples are given in (28) and (29)).

(27) Tagalog perfective infixation (Avery and Lamontagne 1995; Yu 2007)
a. Pattern A: If the stressed syllable is an odd number of syllables from *-in-*, the affix appears after C_1 and before an epenthetic vowel.
b. Pattern B: If the stressed syllable is an even number of syllables from *-in-*, the affix appears after either C_1 or C_2.

(28) Tagalog perfective infixation Pattern A
a. f_{perf}(plahiyó) = p-in-alahiyó ‘plagiarized’
b. f_{perf}(premyuhán) = p-in-iremyuhán ‘rewarded’
c. f_{perf}(plántsa) = p-in-alántsa ‘ironed’

(29) Tagalog perfective infixation Pattern B[19]
a. f_{perf}(prenúhan) = pr-in-enúhan ‘braked’
b. f_{perf}(klipán) = kl-in-ipán ‘cremated’
c. f_{perf}(promót) = pr-in-omót ‘promoted’

The description in (27) of which pattern applies to which word refers to *-in-* itself, which in the context of morphological maps as defined in this paper will only be present in the output form. To recast the distribution in terms of the input, we could revise (27) as follows:

[19] All of these examples have the infix appear after C_2, though the description and data from Avery and Lamontagne (1995), Yu (2007) suggest there is some free variation that has it placed between the two consonants. Free variation cannot be modeled with the deterministic FSTs used throughout this paper, though it may be possible to adapt them to handle variation by making them p-subsequential (Mohri 1997) or semi-deterministic (Beros and de la Higuera 2016).

(30) Tagalog perfective infixation

a. Pattern A: If the stressed syllable is an even number of syllables from the beginning of the word ($\rtimes$), then *-in-* appears after C_1 and before an epenthetic vowel.

b. Pattern B: If the stressed syllable is an odd number of syllables from the beginning of the word, then *-in-* appears after either C_1 or C_2.

Though again not ISL because of the need to track the even/odd parity of the number of syllables before the stressed syllable, this map is subsequential. More specifically, it is right subsequential, meaning the input must be read from right-to-left. Once the stressed syllable is found (starting from the right), the FST can keep track at all times of whether it has seen an even or odd number of additional syllables. When it reaches the end of the string (which in a right subsequential FST would correspond to the start of the word), it will know whether to apply Pattern A (epenthesizing and placing the infix) or Pattern B (placing the infix without epenthesis). The FST will be a bit more complicated than the one in Fig. 17, because the placement of the infix means the consonants can't be ignored in the same way (i.e., those transitions can't just be loops). Instead, there needs to be multiple paths from the 'even' to the 'odd' state (and vice versa), one for each possible syllable type (e.g., CVC, CCVC, CVCC, etc.). Nonetheless, this FST will still be subsequential.

A note of clarification on right subsequential functions. Since string reversal is itself a non-regular operation, then under the assumption that reading the input from right-to-left requires string reversal, the classification of a pattern as right subsequential (and therefore subregular) might appear to be negated by this non-regular preprocessing of the input string. However, right subsequential functions do not literally reverse the input, they simply begin reading it starting from the end instead of the beginning.

This review of affixation maps has already shown some variation, though the nature of this variation is significant in two respects. One, all of the maps surveyed are either ISL for some k or subsequential, meaning they are all subregular. Two, those maps that are properly subsequential all involve conditioning based on metrical and/or prosodic structure. The case of Yidiŋ in particular provides an example of a map whose classification differs depending on whether or not the input is already parsed into feet. A more thorough analysis of the computational nature of metrical parsing and prosodic marking themselves, as well as how these domains interact with phonological and morphological maps, is being left for future work. But these examples suggest that such an investigation may reveal important insights into the morpho-phonological interface.

4.2 Local partial reduplication

Partial reduplication in general involves copying a portion of the base and then affixing that copied material. 'Local' varieties are those in which the location of the affix is adjacent to the material that it was copied from. Two examples are given below. In (31) (again from Tagalog), a CV-prefix is copied from the beginning of the base to derive the future tense of a verb. And in (32) (from Marshallese) a CVC-suffix is copied from the end of the base to derive an adjective from a verb.

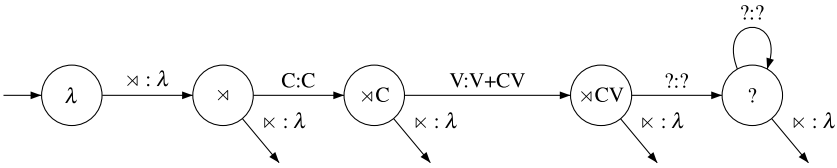


Fig. 18 4-ISL FST for Tagalog reduplicative prefixation

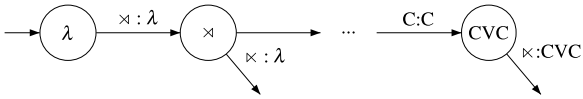


Fig. 19 4-ISL FST for Marshallese reduplicative suffixation

(31) Tagalog (Blake 1917)
f_{fut}(sulat) = su+sulat ('write' → 'will write')

(32) Marshallese (Byrd 1993)
f_{adj}(ebbok) = ebbok+bok ('to make full' → 'puffy')

In (31), the initial CV is copied and prefixed to the base. The FST that models this map needs to retain the initial CV sequence up to the point when it needs to be affixed; a 4-ISL FST can do this easily. Figure 18 is a schematized FST for this map; the complete FST would have states for all possible initial CV sequences. Only the initial portion is shown (up to state ⋊CV, at which point the reduplication has taken place) for ease of reading. All additional states are collapsed to the '?' state.

The Marshallese pattern is also 4-ISL. Recall that in a 4-ISL FST, the only way to be in a given state, such as state *bok*, is if the last three segments of the input are *bok*. Ending in the state, as would be the case for the input *ebbok*, therefore means those are the last three segments of the string. The reduplication can then be achieved straightforwardly using the final output function: the output on the ⋉ transition for all CVC states is that same CVC. This is schematized in the FST in Fig. 19. Again the complete FST would have states for all possible CVC sequences. But no matter what path leads to a state CVC (as indicated by the ...), strings that end in that state will have the final CVC appended as a suffix.

An example of reduplicative infixation comes from Pima (Riggle 2006). The plural of a noun is derived from the singular by copying either (1) the initial C or (2) the initial CV. These options are shown in (33) and (34) below. The copied material is infixed after the first vowel.

(33) Pima
f_{pl}(mavit) = ma+m+vit ('lion' → 'lions')

Considering first just the C-copying variant, the map can be modeled with the schematized FST in Fig. 20.

Riggle (2006) argues that C-infixation is the default pattern and the CV variant occurs to avoid certain consonants in coda position. For example, CV is copied in

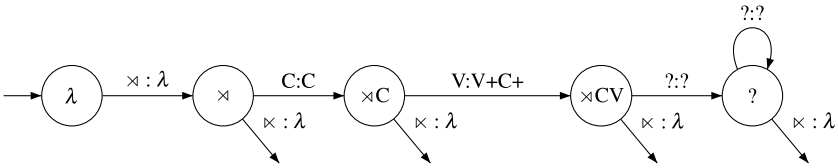


Fig. 20 4-ISL FST for Pima reduplicative infixation (C-copying variant only)

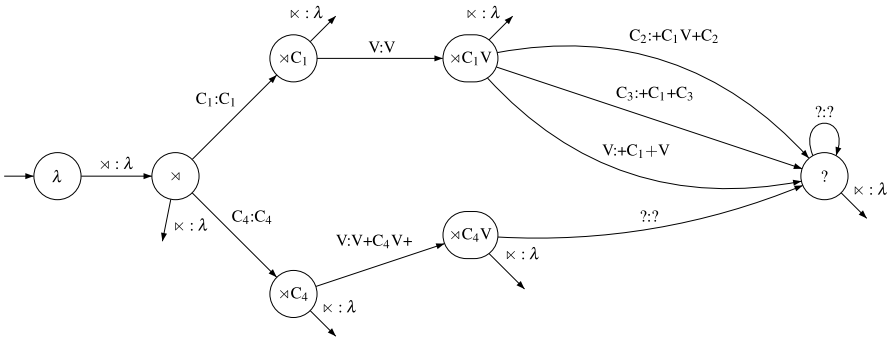


Fig. 21 4-ISL FST for Pima reduplicative infixation

(34) because laryngeals (34a) and palatal nasals (34b) are not preferred as codas.[20] Copying CV instead of C puts these consonants in onset position instead. CV is also copied when copying C would result in a coda cluster with a sonority plateau (34c).[21]

(34) a. f_{pl}(hod̪ai) = ho+ho+d̪ai ('rock' → 'rocks') (*hoh.d̪ai)
b. f_{pl}(ɲumatʃ) = ɲu+ɲu+matʃ ('liver' → 'livers') (*ɲuɲ.matʃ)
c. f_{pl}(gogs) = go+go+gs ('dog'→ 'dogs') (*goggs)

Factoring in these conditions governing the CV-variant we get the FST in Fig. 21. In this FST, C_4 represents consonants that are not permitted in coda position (laryngeals and palatal nasals). When the input string starts with a C_4, the vowel is also copied and infixed: the output of the subsequent V transition is V+C_4V+. All other consonants are represented with C_1. On this branch of the FST, the output that includes the copied material is delayed one segment past the first vowel; this is necessary to handle the restriction on sonority plateaus. The FST uses the next segment after the vowel to determine whether to copy CV or just C. Let C_2 be a consonant that is not less sonorous than C_1 and C_3 be a consonant that is less sonorous than C_1. If the next

[20] Riggle (2006) notes that palatal nasal codas are not banned generally in Pima, just in the context of reduplication.

[21] Riggle (2006) also gives examples of forms with complex onsets, in which the second consonant of the onset copies (along with the vowel according to the generalizations already discussed): klavo ↦ klа+lı+vo, 'nails'. He notes that only a few such forms exist in the language, but the fact that they follow the general pattern suggests that the infixation map should include them. The FST given in the text could easily be modified to handle complex onsets; this would increase the k-value by 1.

Input:		⋊		m		a		v		i		t		⋉				
States:	λ	→	⋊	→	⋊C_1	→	⋊C_1V	→	**?**	→	?	→	?	→				
Output:		λ		m		a		+m+v		i		t		λ				

Fig. 22 Path through FST in Fig. 21 for input *mavit*. Note: the bold ? state was reached via the C_3 transition

Input:		⋊		h		o		d̪		a		i		⋉				
States:	λ	→	⋊	→	⋊C_4	→	⋊C_4V	→	?	→	?	→	?	→				
Output:		λ		h		o+ho+		d̪		a		i		λ				

Fig. 23 Path through FST in Fig. 21 for input *hod̪ai*

Input:		⋊		ɲ		u		m		a		tʃ		⋉				
States:	λ	→	⋊	→	⋊C_4	→	⋊C_4V	→	?	→	?	→	?	→				
Output:		λ		ɲ		u+ɲu+		m		a		tʃ		λ				

Fig. 24 Path through FST in Fig. 21 for input *ɲumatʃ*

Input:		⋊		g		o		g		s		⋉						
States:	λ	→	⋊	→	⋊C_1	→	⋊C_1V	→	**?**	→	?	→						
Output:		λ		g		o		+go+g		s		λ						

Fig. 25 Path through FST in Fig. 21 for input *gogs*. Note the bold ? state was reached via the C_2 transition

segment after the vowel is a C_2, both C_1 and the vowel are copied. Otherwise (if the next segment is a C_3 or a vowel), just C_1 is copied.

The paths through this FST for the examples in (33) and (34) are shown in Figs. 22, 23, 24, 25.

4.3 Non-local partial reduplication

A non-local reduplication map is one in which the affix is copied from one end of the string and then attached to the opposite edge.[22] An example from Chukchee (Bogoras 1969) is shown in (35). The absolute form[23] of a noun is derived by suffixing a copy of the initial CVC sequence.

[22]This type of reduplication has also been called 'wrong side reduplication', and its status is controversial. Nelson (2003) argues that all purported cases are epiphenomenal, while Riggle (2003) and more recently Kusmer and Hauser (2016) argue for genuine examples in Creek/Muskogean and Koasati, respectively. The analysis presented in this paper is not an argument for or against the existence of non-local reduplicative copying; it only reveals the computational properties of such a map.

[23]This is the form used when the noun is an intransitive subject or a transitive object (Bogoras 1969).

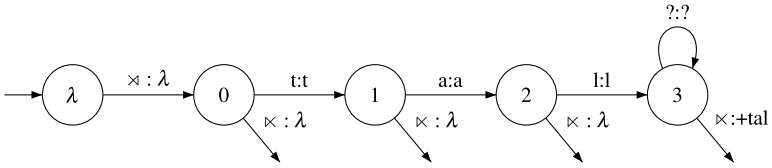


Fig. 26 Fragment of left subsequential FST for Chukchee partial reduplication

(35) Chukchee

a. f_{abs}(nute) = nute+nut 'land'
b. f_{abs}(tala) = tala+tal 'meat'

Non-local reduplication maps are not ISL (or OSL). The reason is because once the end of the string is reached the FST needs to recall how the string began in order to suffix that material onto the end. This is not possible in an ISL/OSL FST. For example, if we set $k = 3$, then for the input *nute* a 3-ISL FST would end in state *te*. From this state, it could not distinguish the input *nute* (for which it would have to suffix *nut*) from a hypothetical input *sute* (for which it would have to instead suffix *sut*). Increasing k to 5 would help, since then *nute* would end in state *nute* and *sute* would end in state *sute*, and each state would have its own final output transition that would append the correct suffixes. But of course this 5-ISL FST would again fail on inputs longer than 4 segments long. As an unprincipled and arbitrary upper bound on the length of words in a given language is an undesirable fix, we conclude that non-local reduplicative suffixation is not ISL for any k.

This map is, however, still subregular, because it can be modeled with a left subsequential FST, a fragment of which is shown in Fig. 26. The complete FST would include 'branches' for all possible CVC sequences; each branch ends in a CVC state where all additional segments are outputted unchanged (via the ?:? self-loop).[24] Once the end of the input is reached, the final output function appends the correct CVC suffix according to the state it is in. The fragment shown in Fig. 26 is the branch of the FST that covers *tala* and all other input strings that begin with *tal*. State 3 is essentially a record of the first three segments that the string began with.

This example shows how the states of a subsequential FST are more flexible than ISL/OSL FSTs in the kind of information they can keep track of. Each CVC state in the Chukchee FST records a distinct initial sequence of the input string. An ISL/OSL FST is more restricted, in that the states can only keep track of the most recent input/output. This is again what makes the ISL/OSL computational property more restrictive, and what limits the kinds of maps that can be represented with ISL/OSL FSTs.

[24]It is important to understand the difference between the ?:? self-loop in Fig. 26 and those in the ISL FSTs in Figs. 16, 18, 20, and 21. In the ISL FSTs, state ? and its self-loop are an abbreviation for the states and transitions for all other $k - 1$ sequences not pictured in the figure. These FSTs proceed through these states depending on the input. In Fig. 26, however, the FST remains in state *tal* and consumes all additional input with the ?:? loop. It must stay in this state to retain the knowledge that the input began with *tal*.

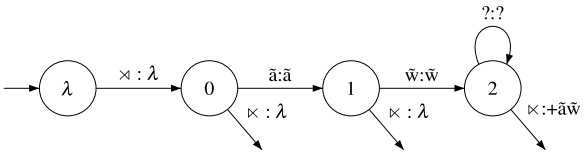


Fig. 27 Fragment of right subsequential FST for Madurese partial reduplication

Input:		⋊		ã		w̃		õ		m		⋉
States:	λ	→	0	→	1	→	2	→	2	→	2	→
Output:		λ		ã		w̃		õ		m		+ãw̃

Fig. 28 Path through FST in Fig. 27 for input *mõw̃ã*

The other option for non-local reduplication is prefixing material copied from the end, as in the Madurese pluralization example in (36) (McCarthy and Prince 1995; Inkelas and Zoll 2005).

(36) Madurese
f_{pl}(mõw̃ã) = w̃ã+mõw̃ã ('face' → 'faces')

This map is right subsequential. The FST in Fig. 27 is the fragment responsible for the map when the input is *mõw̃ã*. In form this FST looks identical to the Chukchee FST, but the crucial difference is how it is applied to an input to generate an output. The string *mõw̃ã* would be read starting from the right, which is equivalent to treating the input as the reversed string *ãw̃õm*. The FST would follow the path shown in Fig. 28. The output, ãw̃õm+ãw̃, is then reversed to the correct surface form: w̃ã+mõw̃ã.[25]

In sum, non-local partial reduplication maps are neither ISL nor OSL, but they are still subregular. Non-local suffixation is left subsequential and non-local prefixation is right subsequential. The next subsection will turn to the analysis of total reduplication, which is quite computationally distinct from all of the morphological maps analyzed thus far.

4.4 Total reduplication

In contrast to partial reduplication, total reduplication—in which the entire string is copied—is not even regular (let alone subregular). An example from Indonesian is given in (37).

(37) Indonesian (Sneddon 1996)
f_{pl}(buku) = buku+buku ('book' → 'books')

Based on the examples of partial reduplication above, at first glance it might seem straightforward to model the map in (37) with an FST. For example, the input string *buku* could be handled easily enough with the FST fragment in Fig. 29.

[25] Again, this does not mean string reversal is required to model the pattern as subsequential. This is just a way to represent the pattern with an FST that reads left-to-right, to be consistent with the other FSTs presented in the paper. In every such case, there is an equivalent FST that reads from the right and also builds the output string starting from the right, such that no string reversal operation is needed.

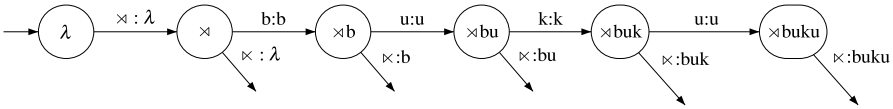


Fig. 29 Fragment of total reduplication FST

Each state in this FST is an exact record of the input that has been read so far. The total reduplication is achieved by simply having the final output function append that same record of input to the end of the string. For example, the input *buku* would end in state ⋊buku, at which point the output is also *buku*. The final transition on ⋉ appends the string +*buku*, so the complete output of the map is *buku*+*buku*. The k-value of this function is one more than the longest state required; in this example $k = 6$.

The problem with this approach by now should be evident. In all previous examples of ISL functions, the value of k was inherent to the nature of the map itself. In Pima reduplicative infixation (Fig. 21), for example, $k = 4$ because the factors that condition the shape of the reduplicant are all found in a contiguous substring bounded by length 4 (e.g., ⋊CV(C)). In Fig. 29, however, k is set to 6 because that happens to be the length of the longest input it needs to deal with (i.e., ⋊buku⋉). So while it's true that an ISL FST can be constructed to correctly apply the total reduplication map to all current Indonesian words, such a FST does not truly represent the map in a fundamental way. Presumably, given a nonsense word that exceeds the length of all current words, an Indonesian speaker would still apply total reduplication correctly. This would suggest that the computations involved in the map are independent of the length of the input. So the needed assumption for modeling total reduplication in the way suggested by Fig. 29—that it is a *finite* map—is incorrect.

Once we treat the map as infinite—meaning it has to handle inputs of any length—there is no way to represent it with a finite number of states. As indicated with the *buku* example, the required number of states is based on the number of words. If the number of words is infinite, so must be the number of states. Again one of the characterizations of regular relations is that they are describable with finite state automata. It follows that a map that is not finite state is not regular.[26]

If, however, the assumption that the reduplication map is infinite is relaxed—meaning it is only defined for a finite set of strings—then there are ways to model it as finite state. Figure 29 points to one approach. Another comes from Roark and Sproat (2007), who build an FST that takes an input string of up to n symbols and generates all possible output strings of $2n$ indexed symbols. For example, the input *buku* would be mapped to a set of strings of the form $X_1X_2X_3X_4$-$b_1u_2k_3u_4$, where X ranges over all symbols in the alphabet. This set is then filtered down to the one string in which all indexed positions match. Since the set includes all possible strings, the correct output *bukubuku* is guaranteed to be in it. Along the same lines, Hulden (2009a) starts with

[26] Engelfriet and Hoogeboom (2001) show that total reduplication can be modeled using graph transductions defined with Monadic Second Order (MSO) logic formulae. This is an interesting result because in terms of formal languages MSO formulae correspond exactly to the regular languages. The total reduplication example, however, proves that the same is not true for maps: MSO formulae can describe both regular and non-regular relations.

a lexicon for which total reduplication applies (e.g., singular nouns) and maps each string x to a set of output strings xy, where y is any singular noun. A regular expression operator is then applied to filter out those strings for which x and y are different nouns (with the result that the only remaining string will be xx).

These treatments of total reduplication as finite state highlight the distinction between just getting the model to 'work' and representing it in a way that has some connection to the way it is represented in the mind of an actual speaker. Though effective, such 'generate and filter' models seem—in an intuitive sense—to do more work than necessary when one envisions how a speaker achieves the map of total reduplication.[27] To reiterate the goals of the current paper, classifying maps in terms of their computational complexity is for the purpose of achieving a greater understanding of why natural language maps are restricted in the ways that they are.[28] To that end, the working assumption has been that these maps are infinite, in which case we are forced to the conclusion that total reduplication is non-regular.

The results presented in this section have followed from the assumption that a reduplication map involves *copying*, as in theories that posit a morpheme whose phonological content is derived by copying from the base it attaches to (e.g., Marantz 1982; McCarthy and Prince 1995). An alternative is Morphological Doubling Theory (Inkelas and Zoll 2005, implemented by Roark and Sproat 2007) in which reduplication is instead the result of two distinct lexical insertion operations. From this perspective, the exceptional nature of total reduplication in being non-regular may in fact support the claim of Heinz and Idsardi (2013) noted above that in terms of computational properties, morpho-phonology patterns with phonology while morpho-syntax patterns with syntax.

4.5 Featural affixation

The examples of affixation surveyed so far have all involved appending segmental material to a string. In featural affixation, one or more features associate to some number of existing segments in the stem. An example is imperfective palatalization in Mafa (Barreteau and Bleis 1990; Ettlinger 2004):

(38) Mafa

a.	f_{imp}(tsap) = tʃep	'is spackling with clay'
b.	f_{imp}(lubat) = lybet	'is twisting'
c.	f_{imp}(sɔban) = ʃiben	'is working'
d.	f_{imp}(gum) = gum	'is carving wood'

[27] This is not a criticism of the works just cited, as they were clearly motivated by different research questions and objectives.

[28] An anonymous reviewer questions this goal, given that computational complexity does not necessarily correspond to the level of processing difficulty (see, e.g., Bach et al. 1986). But there are other areas of interest that the study of computational properties can inform aside from processing, such as evaluating the generative capacity of a particular theory. In addition, computational properties—particularly subregular ones—provide an inroad to understanding how the grammars used in processing are learned in the first place (for arguments in favor of this approach see Heinz 2007, 2009, 2010).

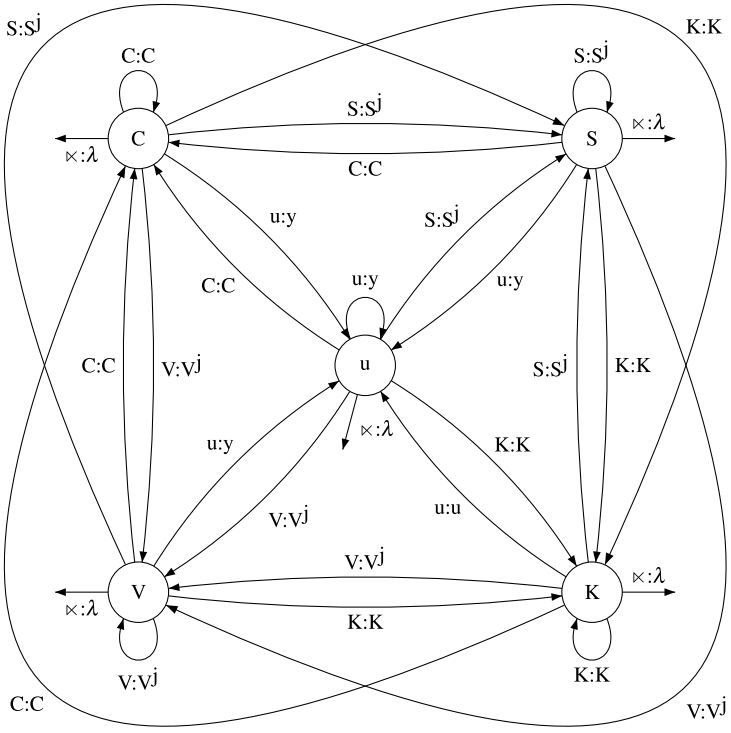


Fig. 30 2-ISL FST for Mafa imperfective palatalization

The segment inventory of Mafa includes five alveolar obstruents that are targeted by palatalization in the imperfective map: ^{n}dz ↦ ^{n}dʒ, s ↦ ʃ, z ↦ ʒ, ts ↦ tʃ, and dz ↦ dʒ.[29] Palatalization also targets the language's four [−palatal] vowels, which then surface as their [+palatal] counterparts: u ↦ y, o ↦ œ, ə ↦ i, and a ↦ e. The generalization for the imperfective is simply that the map palatalizes all segments that have a palatal counterpart, with one exception. As shown in (38d), palatalization of /u/ is blocked after velars, due to a phonotactic constraint against velar+u sequences.

Mafa imperfective palatalization is 2-ISL, as demonstrated with the 2-ISL FST in Fig. 30. Again to keep the FST readable, the following abbreviated alphabet is used: C is any consonant except velars and alveolars, K is any velar, S is any alveolar (with S^{j} being its palatal counterpart), V is any vowel except u (with again V^{j} being its palatal counterpart), and u is itself. The λ and ⋊ states are also not shown for readability; it can be assumed that (as in the previous FSTs), there is a transition from λ to ⋊ and then transitions from ⋊ to all other states. The paths for the inputs /səban/ and /gum/ are shown in Figs. 31 and 32, respectively.

[29] Mafa distinguishes dental and alveolar: the obstruents /t/, /d/, and /^{n}d/ are classified as dental and therefore not subject to palatalization (Ettlinger 2004).

Input:		⋊		s		ə		b		a		n		⋉	
States:	λ	→	⋊	→	S	→	V	→	C	→	V	→	C	→	
Output:		λ		ʃ		i		b		e		n		λ	

Fig. 31 Path through FST in Fig. 30 for input *səban*

Fig. 32 Path through FST in Fig. 30 for input *gum*

Input:		⋊		g		u		m		⋉
States:	λ	→	⋊	→	K	→	u	→	C	→
Output:		λ		g		u		m		λ

Based on this, featural affixation alone appears to be ISL, but the Mafa case presents an additional complication. For stems that end in a vowel, the imperfective morpheme is a segmental suffix, *-j*:

(39) Mafa
f_{imp}(gudza) = gudza+*j* 'is trembling'

There are two ways to address this additional fact. One approach is to posit two distinct imperfective maps, one for stems ending in vowels and one for stems ending in consonants. Stems ending in consonants would be input to the 2-ISL map presented in this section, and stems ending in vowels would be input to a 1-ISL suffixation map like the *ɪŋ*-suffixation example in Fig. 9. Under this approach, it would be a mere coincidence that both the featural and segmental versions of the imperfective affix are palatal. The alternative is to generate both types of the imperfective by a single map, in the spirit of the OT-based analysis of Ettlinger (2004).

In this latter approach, however, the map is no longer ISL. The map has to palatalize all palatalizable segments *unless* the input ends in a vowel, in which case it just appends *-j*. The need to retain information about the end of the string while processing the rest of the string violates the essential notion of Strict Locality that defines ISL/OSL maps.The map is, however, still subregular: it is right subsequential, as evidenced by the right subsequential FST in Fig. 33.

After reading the first segment, this FST determines whether it should perform suffixation (in which case it proceeds to state 3) or palatalization (in which case it proceeds to state 1). Suffixation is performed on the transition to state 3, where all additional segments are then outputted unchanged.[30] From state 1, however, palatalization proceeds much as it did in Fig. 30; the blocking of u ↦ y after velars is handled by an additional state, 2.

The Mafa case thus provides an example of featural affixation (which is ISL), but also shows how certain types of allomorphy—in this case the combination of featural and segmental affixation—can increase the computational complexity of a map. More will be said about combining maps in Sect. 5. The next section will turn to another type of non-concatenative morphology, truncation.

[30]Recall that in a right subsequential FST, the input is read from the right and the resulting output string is reversed. Thus the input /gudza/ would be read as /azdug/, and the resulting output, [j+azdug], would be reversed to [gudza+j].

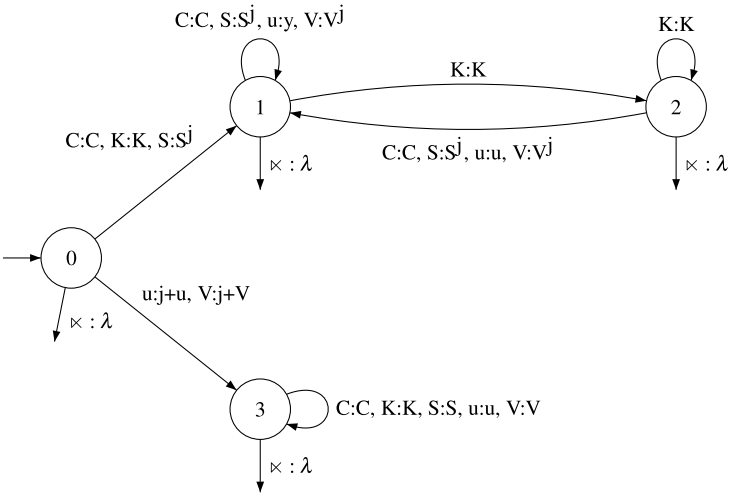


Fig. 33 Right subsequential FST for Mafa imperfective palatalization and suffixation

4.6 Truncation

A simple case of morphological truncation is English nickname formation, a map that deletes all but the first (C)(C)VC of a name (Inkelas and Zoll 2005). Examples of this map are given in (40).

(40) English

a. $f_{nn}(\text{dʒɛfɹi}) = \text{dʒɛf}$
b. $f_{nn}(\text{deɪvɪd}) = \text{deɪv}$
c. $f_{nn}(\text{ælɛn}) = \text{æl}$
d. $f_{nn}(\text{stivɪm}) = \text{stiv}$

This map is left OSL, as evidenced by the 3-LOSL FST in Fig. 34. This FST outputs the initial sequence of the input up to and including the first VC sequence, at which point it remains in state VC and deletes all additional input with self-loops that have λ as the output. These self-loops are why the FST is necessarily OSL instead of ISL. Recall that an OSL FST at any given point is in the state that represents the most recent *output*. If a transition outputs λ, then the recent output has not changed and therefore the FST must remain in the state it was already in (i.e., the transition must be a loop). In the case of truncation, this is exactly what is needed: all additional input after the first VC is deleted.

4.7 Summary

Table 1 summarizes the computational classifications of the morphological maps reviewed in this section. The next section will discuss the implications of these results and offer some comments on two notable omissions from this table: templatic morphology and compounding.

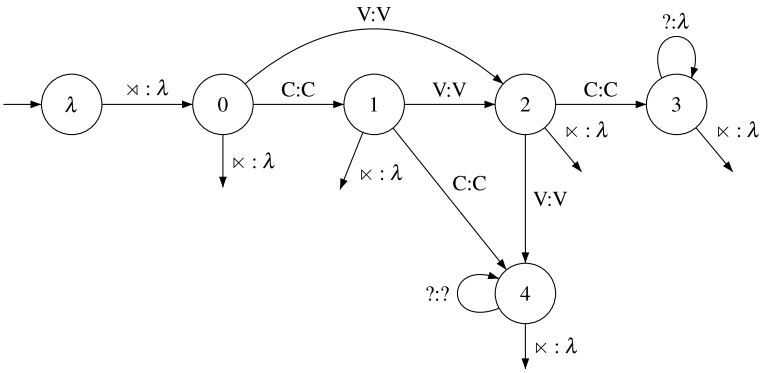


Fig. 34 3-LOSL FST for English nickname formation

Table 1 Summary of morphological maps and their computational classification

Map	Language	Classification
Suffixation	English	1-ISL
Suffixation	Sammi	left subsequential
Suffixation	Yidiŋ	left subsequential
Suffixation	Yidiŋ	7-ISL (with foot structure)
Prefixation	English	1-ISL
Circumfixation	Chickasaw	1-ISL
Circumfixation	Chickasaw	2-ISL (with deletion)
Circumfixation	German	3-ISL
Infixation	Tagalog (*um*)	4-ISL
Infixation	Ulwa	6-ISL
Infixation	Tagalog (*in*)	right subsequential
Reduplicative prefixation (local)	Tagalog	4-ISL
Reduplicative suffixation (local)	Marshallese	4-ISL
Reduplicative infixation	Pima	4-ISL
Reduplicative prefixation (non-local)	Madurese	right subsequential
Reduplicative suffixation (non-local)	Chukchee	left subsequential
Total reduplication	Indonesian	non-regular
Featural affixation	Mafa (palatalization)	2-ISL
Featural affixation	Mafa (imperfective)	right subsequential
Truncation	English	4-LOSL

5 Discussion

This paper has presented a substantial—but not complete—catalog of the computational nature of morphological maps. This section discusses (1) a couple of remaining operations that await a thorough analysis and (2) the implications of these results.

The most prominent type of morphological map for which the computational properties are as yet unknown is templatic morphology, in which a particular morphological form is represented with a template of C and V slots that are filled in by particular consonants and vowels depending on the lexical item. A classic example from Arabic (McCarthy 1981) is shown in (41).

(41) Arabic

a. kataba 'he wrote'
b. kattaba 'he caused to write'
c. kutiba 'it was written'

The main question for treating templatic morphology as a map is to determine what the input to the map would be. There are typically assumed to be three components to these derivations (McCarthy 1981): the consonant root (*ktb*), the vowel melody (*a*, *ui*), and the prosodic template (CVCVCV, CVCCVCV, etc.). The most direct translation of these facts to the concept of a map would be a function that takes three inputs and produces an output, as in (42).

(42) $f(\text{ktb, a, CVCVCV}) = \text{kataba}$

However, the framework employed in this paper does not provide any such class of functions. Another option would be for two functions to represent the consonant root and the vowel melody, with the template being the input to both in turn. Each function fills in its respective slots on the template. This is shown in (43). These two functions can be combined into a single function via composition, as in (43c).

(43) a. $f_{ktb}(\text{CVCVCV}) = \text{kVtVbV}$
b. $f_{a}(\text{kVtVbV}) = \text{kataba}$
c. $f_{a} \circ f_{ktb}(\text{CVCVCV}) = \text{kataba}$

The problem with this approach is that we lose the association of a map to a morphological operation, since the function now represents a lexical item and the input represents the operation (i.e., 'past tense' is input to the function 'write'). Really what we want is something like (44), in which the template is the map.

(44) a. $f_{past1}(\text{ktb}) = \text{kVtVbV}$
b. $f_{past2}(\text{a}) = \text{CaCaCa}$

But these two maps cannot be combined via composition, since the input to the second one is not the output of the first. Previous approaches to modeling templatic morphology as finite state have therefore had to modify the general approach that worked for other types of operations. Beesley and Karttunen (2003) define a new operation for combining maps called 'compile and replace' and use it instead of composition.[31] Roark and Sproat (2007) start with the assumption that the template includes the vowel melody (e.g., CaCaC), along the lines of Harris (1941). And Hulden (2009b)

[31] They also use this operation in their treatment of reduplication, see Beesley and Karttunen (2003) for details.

employs a multi-tape automaton with 8 tapes instead of the standard 2.[32] More work is needed, then, to identify the right method of representing templatic morphology as a map with the goal of understanding its computational properties.

Two additional morphological operations to consider are suppletion and ablaut. Suppletion is not really a map at all, since a form like *went* is not derived from an input like *go* by any operation. However, suppletion could be viewed as a *finite* map, under the assumption that suppletive forms in a language are fixed (i.e., non-productive). Lastly, ablaut is a simple substitution operation (like many phonological maps). Depending on the language, it may again be a finite map (if it is not a productive pattern). Alternatively, if the context in which ablaut occurs can be generalized and represented as a contiguous substring of bounded length, it is ISL. A systematic survey of both suppletion and ablaut would be needed to fully test these assumptions, though a reasonable conjecture is that both operations are subregular.

Lastly, an anonymous reviewer asks how the operation of compounding fits into this picture. Again the question is how to construe the operation as a map. In an example like 'desk chair', this would involve mapping two strings to a single string via concatenation. As with templatic morphology above, it is unclear how to classify such a function using the methods employed in this paper, and so the right way to think of compounding as a map merits further consideration. This complication in how to conceive of compounding as a map does draw attention to the ways in which it differs from other operations. For example, Carden (1983) notes that perhaps compounding is achieved in the syntactic component rather than the word formation component.

Turning now to the second question: what do we do with these results? The motivation for undertaking such analyses in the first place is the same as that argued for phonology in previous work (Heinz 2007, 2009, 2010; Chandlee 2014; Jardine 2016): understanding the computational nature of a set of patterns is a significant inroad to understanding what that set is, and—equally importantly—what it is not. Human languages have shown both variability and creativity in how to encode morphological distinctions. But that variability is not endless and, as with phonology, we can entertain logically possible morphological maps that no known language takes advantage of. For example, a language could represent the plural form of a noun with the operation of string reversal, as in (45).

(45) "English", singular → plural
  a. /kæt/ → [tæk]

Such a map is computable and relatively simple (one could imagine a language game that follows such a principle), but from a linguistic perspective it is an odd morphological operation and indeed appears to be unattested. The computational approach advocated in this paper provides an answer in terms of computational restriction: there is strong evidence that morphological maps, like phonological ones, are restricted to being subregular. String reversal is not.[33]

[32] The 'tapes' of an automaton refer to the number of strings being read or written. In all of the FSTs in this paper one tape corresponds to the input and one to the output.

[33] An anonymous reviewer points out that this operation would be regular if the reversal were bounded. But, as with total reduplication, the assumption is that the domain of the function is any possible noun (i.e., it's unbounded).

These analyses have also raised interesting questions about what constitutes a morphological generalization and how generalizations can interact. The analysis of total reduplication in Sect. 4.4 highlighted how certain assumptions about the nature of a given map—whether it is finite or infinite and whether it involves copying—have real implications for its computational classification. And the Mafa case in Sect. 4.5 demonstrated the consequences for computational properties when a set of facts is analyzed as one map or two distinct maps. This case was not one of ordering as is common for map interactions in phonology, but rather a type of disjunction based on a partition of the domain (words that end in consonants and words that end in vowels). How such an interaction fits into our larger understanding of how individual generalizations can interact in the grammar is an intriguing and ongoing line of inquiry (see Baković 2013). In this way the analytical framework employed here can bring to light questions of larger theoretical interest regarding morphological operations and the morpho-phonological interface.

6 Conclusion

This paper has contributed to the foundation for computational investigations into natural language morphology. It was shown that a variety of morphological operations—analyzed as morphological maps from an input to an output—share computational properties with phonological maps in being subregular and therefore less computationally complex than syntax. This does not, however, amount to a claim that morphology in its entirety is subregular. As was discussed—and as was already known—the non-regular status of total reduplication bars such a conclusion. Rather, the results collectively show that in this regard total reduplication is something of an outlier, as all of the other operations analyzed were not only regular, but subregular.

In addition, the classifications summarized in Table 1 also provide a more detailed and nuanced view of the notion of computational locality that formed the backbone of previous analyses in phonology (Heinz 2009, 2010; Chandlee 2014; Jardine 2016). The designation of what phonological maps are ISL/OSL is stated in terms of the target and triggering context, but it is not always clear how this extends to morphological maps. For example, what is the triggering context of prefixation? The more general statement is that maps—be they phonological or morphological—are Strictly Local provided the crucial information needed at any given time to determine the output is a bounded number of segments away. While this notion of locality appears to be prominent among morphological maps, there were also several exceptions that raised significant questions for our understanding of what various domains (phonology, morphology, syntax) are responsible for and what happens computationally when these domains interact.

The hope is that these results pave the way for further investigation, not only to fill in the gaps in the typology (e.g., templatic morphology), but also to examine the assumptions that some of the analyses relied on. Such assumptions are meant to reflect the nature of the morphological operation itself, but there were several cases where alternative assumptions led to a different computational classification. In this way the identification of computational properties can highlight what's at stake for competing analyses and interpretations of natural language patterns in various domains.

Appendix

For ease of exposition the analysis of German circumfixation in Sect. 4.1 treated separately the two generalizations for the distribution of the suffix allomorph *-et*. This appendix presents the complete 3-ISL FST that models the allomorphy as a single map. The FST is presented in table form for readability. Each row q_1 of the table corresponds to a state in the FST, and each column a corresponds to one of the input segments that can be read from that state. The table cells contain pairs (b, q_2) where b is the output produced for input a and q_2 is the destination state of the transition from q_1 for input a. To further illustrate how this table representation corresponds to the graphical representations used throughout the paper, those transitions represented graphically in Fig. 15 are shaded in the table. The alphabet is $\Sigma = \{\text{L, N, T, ?}\}$, where '?' represents all segments in the German inventory except for liquids, nasals, and alveolar stops.

	⋊	T	?	L	N	⋉
λ	(ge+, ⋊)	–	–	–	–	–
⋊	–	(T, T)	(?, ?)	(L, L)	(N, N)	+t
L	–	(T, LT)	(?, L?)	(L, LL)	(N, LN)	+t
N	–	(T, NT)	(?, N?)	(L, NL)	(N, NN)	+et
T	–	(T, TT)	(?, T?)	(L, TL)	(N, TN)	+et
?	–	(T, ?T)	(?, ??)	(L, ?L)	(N, ?N)	+t
LL	–	(T, LT)	(?, L?)	(L, LL)	(N, LN)	+t
LN	–	(T, NT)	(?, N?)	(L, NL)	(N, NN)	+t
LT	–	(T, TT)	(?, T?)	(L, TL)	(N, TN)	+et
L?	–	(T, ?T)	(?, ??)	(L, ?L)	(N, ?N)	+t
NL	–	(T, LT)	(?, L?)	(L, LL)	(N, LN)	+t
NN	–	(T, NT)	(?, N?)	(L, NL)	(N, NN)	+et
NT	–	(T, TT)	(?, T?)	(L, TL)	(N, TN)	+et
N?	–	(T, ?T)	(?, ??)	(L, ?L)	(N, ?N)	+t
TL	–	(T, LT)	(?, L?)	(L, LL)	(N, LN)	+t
TN	–	(T, NT)	(?, N?)	(L, NL)	(N, NN)	+et
TT	–	(T, TT)	(?, T?)	(L, TL)	(N, TN)	+et
T?	–	(T, ?T)	(?, ??)	(L, ?L)	(N, ?N)	+t
?L	–	(T, LT)	(?, L?)	(L, LL)	(N, LN)	+t
?N	–	(T, NT)	(?, N?)	(L, NL)	(N, NN)	+et
?T	–	(T, TT)	(?, T?)	(L, TL)	(N, TN)	+et
??	–	(T, ?T)	(?, ??)	(L, ?L)	(N, ?N)	+t

References

Anderson, S. R. (1992). *A-morphous morphology*. Cambridge: Cambridge University Press.

Aronoff, M. (1994). *Morphology by itself. Stems and inflectional classes*. Cambridge: MIT Press.
Avery, P., & Lamontagne, G. (1995). *Infixation <and metathesis> in Tagalog*. Paper presented at the Canadian Linguistics Association, Montreal, 3 June.
Bach, E., Brown, C., & Marslen-Wilson, W. (1986). Crossed and nested dependencies in German and Dutch: a psycholinguistic study. *Language and Cognitive Processes*, *1*(4), 249–262.
Baković, E. (2013). *Blocking and complementarity in phonological theory*. Bristol: Equinox.
Bar-Hillel, Y., & Shamir, E. (1960). Finite-state languages: formal representations and adequacy problems. *Bulletin of the Research Council of Israel*, *8F*, 155–166. Reprinted in Y. Bar-Hillel (1964) Language and Information, Addison-Wesley, Reading, Massachusetts.
Barreteau, D., & Bleis, Y. L. (1990). *Lexique Mafa*. Paris: Librairie Orientaliste Paul Geuthner.
Beesley, K. R., & Karttunen, L. (2003). *Finite state morphology*. Stanford: Center for the Study of Language and Information.
Bergsland, K. (1976). *Lappische Grammatik mit Lesestücken*. Wiesbaden: Otto Harrassowitz.
Beros, A., & de la Higuera, C. (2016). A canonical semi-deterministic transducer. *Fundamenta Informaticae*, *146*(4), 431–459.
Blake, F. R. (1917). Reduplication in Tagalog. *The American Journal of Philology*, *38*(4), 425–431.
Bogoras, W. (1969). Chukchee. In F. Boas (Ed.), *Bureau of American ethnology bulletin: Vol. 40. Handbook of American Indian languages, Part 2* (pp. 631–903). Washington: Government Printing Office.
Bromberger, S., & Halle, M. (1988). *Conceptual issues in morphology. Ms.*, Cambridge: MIT Press.
Bromberger, S., & Halle, M. (1989). Why phonology is different. *Linguistic Inquiry*, *20*, 51–70.
Byrd, D. (1993). Marshallese suffixal reduplication. In J. Mead (Ed.), *WCCFL 11: proceedings of the 11th West coast conference on formal linguistics* (pp. 61–77).
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.
Chandlee, J., & Heinz, J. (2012). Bounded copying is subsequential: implications for metathesis and reduplication. In *Proceedings of the twelfth meeting of the special interest group on computational morphology and phonology (SIGMORPHON2012)* (pp. 42–51). Chicago: Association for Computational Linguistics.
Chandlee, J., & Heinz, J. (2018). Strict locality and phonological maps. *Linguistic Inquiry*.
Chandlee, J., Athanasopoulou, A., & Heinz, J. (2012). Evidence for classifying metathesis patterns as subsequential. In J. Choi, E. A. Hogue, J. Punske, D. Tat, J. Schertz, & A. Trueman (Eds.), *WCCFL 29: proceedings of the 29th West coast conference on formal linguistics* (pp. 303–309). Somerville: Cascadilla.
Chandlee, J., Heinz, J., & Eyraud, R. (2014). Learning strictly local subsequential functions. *Transactions of the Association for Computational Linguistics*, *2*, 491–503.
Chandlee, J., Eyraud, R., & Heinz, J. (2015). Output strictly local functions. In *Proceedings of the 14th meeting on the mathematics of language (MoL 2015)* (pp. 112–125). Chicago: Association for Computational Linguistics.
Chomsky, N. (1956). Three models for the description of language. *I.R.E. Transactions on Information Theory*, *2*(3), 113–124.
Chomsky, N., & Halle, M. (1968). *The sound pattern of English*. New York: Harper & Row.
Culy, C. (1985). The complexity of the vocabulary of Bambara. *Linguistics and Philosophy*, *8*(3), 345–351.
Daly, R. (1974). *Applications of the mathematical theory of linguistics*. The Hague: Mouton.
Dixon, R. M. W. (1977). *A Grammar of Yidiɲ*. Cambridge: Cambridge University Press.
Engelfriet, J., & Hoogeboom, H. J. (2001). MSO definable string transductions and two-way finite-state transducers. *ACM Transactions on Computational Logic*, *2*(2), 216–254.
Ettlinger, M. (2004). Aspect in Mafa: an intriguing case of featural affixation. In *Proceedings from the annual meeting of the Chicago linguistic society* (Vol. 40, pp. 73–86).
French, K. M. (1988). *Insights into tagalog: reduplication, infixation, and stress from nonlinear phonology*. Arlington: Summer Institute of Linguistics and University of Texas at Arlington.
Fromkin, V., Rodman, R., & Hyams, N. (2014). *An introduction to language* (10 ed.). Belmont: Wadsworth Cengage.
Gainor, B., Lai, R., & Heinz, J. (2012). Computational characterizations of vowel harmony patterns and pathologies. In J. Choi, E. A. Hogue, J. Punske, D. Tat, J. Schertz, & A. Trueman (Eds.), *WCCFL 29: proceedings of the 29th West coast conference on formal linguistics* (pp. 63–71). Somerville: Cascadilla.

Gazdar, G., & Pullum, G. K. (1985). Computationally relevant properties of natural languages and their grammars. *New Generation Computing*, *3*(3), 273–306.
Hale, K., & Blanco, A. L. (1989). *Diccionario elemental del Ulwa (Sumu Meridional)*. Cambridge: Center for Cognitive Science, MIT.
Halle, M., & Marantz, A. (1993). Distributed morphology and the pieces of inflection. In K. Hale & S. J. Keyser (Eds.), *The view from building 20: essays in linguistics in honor of Sylvain Bromberger* (pp. 111–176). Cambridge: MIT Press.
Hansson, G. (2001). *Theoretical and typological issues in consonant harmony*. Ph.D. thesis, University of California, Berkeley.
Hargus, S. (1993). Modeling the phonology-morphology interface. In S. Hargus & E. M. Kaisse (Eds.), *Phonetics and phonology: studies in lexical phonology* (Vol. 4, pp. 45–74). San Diego: Academic Press.
Harris, Z. (1941). Linguistic structure of Hebrew. *Journal of the American Oriental Society*, *61*(3), 143–167.
Hayes, B. (1982). Metrical structure as the organizing principle of Yidiŋ phonology. In H. van der Hulst & N. Smith (Eds.), *The structure of phonological representations, part I*, Dordrecht: Foris Publications.
Hayes, B. (1999). Phonological restructuring in Yidiŋ and its theoretical consequences. In B. Hermans & M. Oostendorp (Eds.), *The derivational residue in phonological optimality theory* (pp. 175–205). Amsterdam: John Benjamins.
Heinz, J. (2007). *The inductive learning of phonotactic patterns*. Ph.D. thesis, University of California, Los Angeles.
Heinz, J. (2009). On the role of locality in learning stress patterns. *Phonology*, *26*, 303–351.
Heinz, J. (2010). Learning long-distance phonotactics. *Linguistic Inquiry*, *41*(4), 623–661.
Heinz, J., & Idsardi, W. (2011). Sentence and word complexity. *Science*, *333*(6040), 295–297.
Heinz, J., & Idsardi, W. (2013). What complexity differences reveal about domains in language. *Topics in Cognitive Sciences*, *5*, 111–131.
Heinz, J., & Lai, R. (2013). Vowel harmony and subsequentiality. In A. Kornai & M. Kuhlmann (Eds.), *Proceedings of the 13th meeting on the mathematics of language (MoL 13)* (pp. 52–63).
Heinz, J., Rawal, C., & Tanner, H. G. (2011). Tier-based Strictly Local constraints for phonology. In *Proceedings of the 49th annual meeting of the association for computational linguistics* (pp. 58–64). Chicago: Association for Computational Linguistics.
Hockett, C. F. (1954). Two models of grammatical description. *Word*, *10*, 210–234.
Hopcroft, J. E., Motwani, R., & Ullman, J. D. (2000). *Introduction to automata theory, languages, and computation*. Reading: Addison-Wesley.
Hulden, M. (2009a). *Finite-state machine construction methods and algorithms for phonology and morphology*. Ph.D. thesis, University of Arizona.
Hulden, M. (2009b). Foma: a finite-state compiler and library. In *Proceedings of the 12th conference of the European chapter of the Association for Computational Linguistics* (pp. 29–32). Chicago: Association for Computational Linguistics. https://code.google.com/archive/p/foma/.
Inkelas, S., & Zoll, C. (2005). *Reduplication: doubling in morphology*. Cambridge: Cambridge University Press.
Jardine, A. (2016). *Locality and non-linear representations in tonal phonology*. Ph.D. thesis, University of Delaware.
Jardine, A., Chandlee, J., Eyraud, R., & Heinz, J. (2014). Very efficient learning of structured classes of subsequential functions from positive data. In A. Clark, M. Kanazawa, & R. Yoshinaka (Eds.), *Proceedings of the twelfth international conference on grammatical inference (ICGI 2014)* (Vol. 34, pp. 94–108). JMLR: Workshop and Conference Proceedings.
Johnson, C. (1972). *Formal aspects of phonological description*. The Hague: Mouton.
Kaplan, R., & Kay, M. (1994). Regular models of phonological rule systems. *Computational Linguistics*, *20*, 371–387.
Kobele, G. (2006). *Generating copies: an investigation into structural identity in language and grammar*. Ph.D. thesis, UCLA.
Koskenniemi, K. (1983). *Two-level morphology: a general computational model for word-form recognition and production*. University of Helsinki, Department of General Linguistics.
Kusmer, L., & Hauser, I. (2016). *Wrong-side reduplication in Koasati*. Paper presented at the 24th Manchester Phonology Meeting.
Langendoen, D. T. (1981). The generative capacity of word-formation components. *Linguistic Inquiry*, *12*, 320–322.

Legendre, G., Miyata, Y., & Smolensky, P. (1990). Harmonic grammar: a formal multi level connectionist theory of linguistic well formedness: theoretical foundations. In *Proceedings of the twelfth annual conference of the Cognitive Science Society*, Cambridge, MA (pp. 388–395).

Luo, H. (2013). *Long-distance consonant harmony and subsequentiality*. Unpublished manuscript.

Marantz, A. (1982). Re reduplication. *Linguistic Inquiry*, *13*(3), 435–482.

McCarthy, J. J. (1981). A prosodic theory of nonconcatenative morphology. *Linguistic Inquiry*, *12*(3), 373–418.

McCarthy, J. J. (2000). Harmonic serialism and parallelism. In M. Hirotani, A. Coetzee, N. Hall, & J. Kim (Eds.), *NELS 30: proceedings of the 30th annual meeting of the North East Linguistic Society* (pp. 501–524). Amherst: GLSA.

McCarthy, J. J., & Prince, A. (1993). *Prosodic morphology I: constraint interaction and satisfaction*. Available at: http://works.bepress.com/john_j_mccarthy/53/.

McCarthy, J. J., & Prince, A. (1995). Faithfulness and reduplicative identity. In J. Beckman, L. Dickey, & S. Urbanczyk (Eds.), *University of Massachusetts Occasional Papers in Linguistics: Vol. 18. Papers in optimality theory* (pp. 249–384). Amherst: GLSA.

McCarthy, J. J., & Prince, A. (1996). *Prosodic morphology 1986*. Available at: http://works.bepress.com/john_j_mccarthy/54/.

McMullin, K. (2016). *Tier-based locality in long-distance phonotactics: learnability and typology*. Ph.D. thesis, University of British Columbia.

McNaughton, R., & Papert, S. (1971). *Counter-free automata*. Cambridge: MIT Press.

Meinhof, C. (1932). *Introduction to the phonology of the Bantu languages*. Berlin: Dietrich Reimer/Ernst Vohsen. Trans. by N. J. van Warmelo.

Mohri, M. (1997). Finite-state transducers in language and speech processing. *Computational Linguistics*, *23*, 269–311.

Mohri, M., & Sproat, R. (2006). On a common fallacy in computational linguistics. In M. Suominen, A. Arppe, A. Airola, O. Heinämäki, M. Miestamo, U. Määttä, J. Niemi, K. K. Pitkänen, & K. Sinnemäki (Eds.), *SKY Journal of Linguistics: Vol. 19. A man of measure: Festschrift in Honour of Fred Karlsson on this 60th Birthday* (pp. 432–439).

Nelson, N. A. (2003). *Asymmetric anchoring*. Ph.D. thesis, Rutgers.

Odden, D. (1994). Adjacency parameters in phonology. *Language*, *70*, 289–330.

Oncina, J., García, J., & Vidal, E. (1993). Learning subsequential transducers for pattern recognition interpretation tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *15*(5), 448–457.

Onn, F. M. (1980). *Aspects of Malay phonology and morphology: a generative approach*. Kuala Lumpur: Universiti Kebangsaan Malaysia.

Orgun, C. O., & Sprouse, R. L. (1999). From mparse to control: deriving ungrammaticality. *Phonology*, *16*, 191–224.

Pater, J. (2012). Serial harmonic grammar and Berber syllabification. In T. Borowsky, S. Kawahara, T. Shinya, & M. Sugahara (Eds.), *Prosody matters: essays in honor of Elisabeth O. Selkirk* (pp. 43–72). London: Equinox.

Payne, A. (2017). All dissimilation is computationally subsequential. *Phonological Analysis*.

Prince, A., & Smolensky, P. (2004). *Optimality theory: constraint interaction in generative grammar*. Oxford: Blackwell.

Riggle, J. (2003). Nonlocal reduplication. In *Proceedings of the 34th annual meeting of the North Eastern Linguistic Society*.

Riggle, J. (2006). Infixing reduplication in Pima and its theoretical consequences. *Natural Language and Linguistic Theory*, *24*(3), 857–891.

Roark, B., & Sproat, R. (2007). *Computational approaches to morphology and syntax*. London: Oxford University Press.

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

Rogers, J., Heinz, J., Fero, M., Hurst, J., Lambert, D., & Wibel, S. (2013). Cognitive and sub-regular complexity. In G. Morrill & M.-J. Nederhof (Eds.), *Lecture notes in computer science: Vol. 8036. Formal grammar* (pp. 90–108). Berlin: Springer.

Rose, S., & Walker, R. (2004). A typology of consonant agreement as correspondence. *Language*, *80*(3), 475–531.

Sapir, E., & Hoijer, H. (1967). *The phonology and morphology of the Navajo language*. *University of California publications in linguistics: Vol. 50*. Berkeley: University of California Press.

Shieber, S. (1985). Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, *8*, 333–343.
Sneddon, J. (1996). *Indonesian: a comprehensive grammar*. London: Routledge.
Sproat, R. (1992). *Morphology and computation*. Cambridge: MIT Press.
Stump, G. (2001). *Inflectional morphology: a theory of paradigm structure*. Cambridge: Cambridge University Press.
Tesar, B. (2008). *Output-driven maps*. ROA-956.
Tesar, B. (2014). *Output-driven phonology: theory and learning*. Cambridge: Cambridge University Press.
Yu, A. C. L. (2007). *A natural history of infixation*. London: Oxford University Press.