Computational Perspectives on Phonological Constituency and Recursion

Item Type	Article
Authors	Yu, Kristine
DOI	10.5565/rev/catjl.354
Rights	Attribution-NonCommercial 4.0 International
Download date	2026-03-10 09:13:11
Item License	http://creativecommons.org/licenses/by-nc/4.0/
Link to Item	https://hdl.handle.net/20.500.14394/32469

Catalan Journal of Linguistics 20, 2021 **77-114**

Computational Perspectives on Phonological Constituency and Recursion*

Kristine M. Yu
University of Massachusetts Amherst. Department of Linguistics
krisyu@linguist.umass.edu

Received: June 30, 2021
Accepted: September 24, 2021

Abstract

Whether or not phonology has recursion is often conflated with whether or not phonology has strings or trees as data structures. Taking a computational perspective from formal language theory and focusing on how phonological strings and trees are built, we disentangle these issues. We show that even considering the boundedness of words and utterances in physical realization and the lack of observable examples of potential recursive embedding of phonological constituents beyond a few layers, recursion is a natural consequence of expressing generalization in phonological grammars for strings and trees. While prosodically-conditioned phonological patterns can be represented using grammars for strings, e.g., with bracketed string representations, we show how grammars for trees provide a natural way to express these patterns and provide insight into the kinds of analyses that phonologists have proposed for them.

Keywords: prosody; recursion; computational phonology; tree transducers

Resum. *Perspectives computacionals de la constituència fonològica i de la recursivitat*

Que la fonologia mostri o no recursivitat sovint va lligat al fet que tingui o no cadenes o arbres en l'estructura de les seves dades. A partir de la perspectiva computacional de la teoria formal del llenguatge i tenint en compte com es construeixen les cadenes i els arbres fonològics, mirem de destriar aquestes qüestions. Mostrem que, fins i tot tenint en compte la limitació de paraules i enunciats en la realització física i la manca d'exemples observables d'incorporació recursiva potencial de constituents fonològics més enllà d'unes poques capes, la recursivitat és una conseqüència natural de l'expressió de generalitzacions fonològiques per a cadenes i arbres. Tot i que els patrons fonològics condicionats prosòdicament es poden representar utilitzant gramàtiques per a cadenes, per exemple amb representacions amb claudàtors, mostrem com les gramàtiques amb arbres proporcionen una manera natural d'expressar aquests patrons i proporcionen coneixement rellevant sobre els tipus d'anàlisis d'aquests patrons que s'han proposat des de la fonologia.

Paraules clau: prosòdia; recursivitat; fonologia computacional; transductors arboris

* This paper builds on a talk with the same title, presented at RecPhon 2019: Recursivity in phonology below and above the word, in November 2019 at Universitat Autònoma de Barcelona, Bellaterra. Many thanks to the conference organizers and presenters for inspiring this work, especially Francesc Torres-Tamarit for sheperding this manuscript through the submission process. Thank you to Ed Stabler for inspiring discussions that helped shape this work and to an anonymous reviewer whose suggestions helped me greatly improve the manuscript.

ISSN 1695-6885 (in press); 2014-9718 (online) https://doi.org/10.5565/rev/catjl.354

Table of Contents

1. Introduction
2. Self-embedding in building phonological strings
3. Self-embedding in building prosodic trees
4. Conclusion
References

1. Introduction

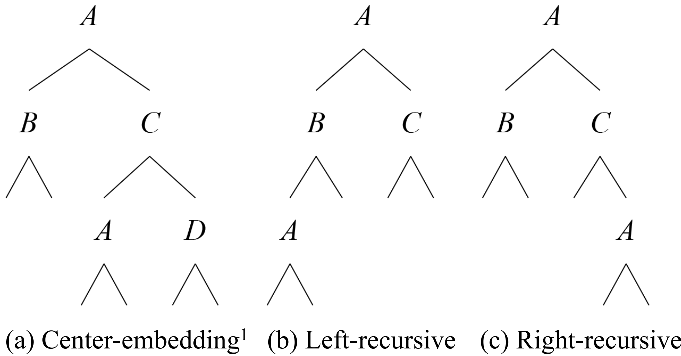
The question of whether or not phonology has recursion is often conflated with the question of whether or not phonology has strings or trees as data structures. This question of strings or trees, in turn, is often linked to the question of whether or not there are such things as phonological constituents. This paper disentangles these issues and shows that once they are factored out and considered from a derivational perspective, it becomes uncontroversial that: (i) phonological grammar is defined over trees in addition to strings, (ii) phonological grammar has recursive structures, if it is acknowledged that there is no principled upper bound on the length of a word, (iii) prosodic tree representations assumed by phonologists—even those that satisfy the Strict Layer Hypothesis (Selkirk 1984; Nespor & Vogel 1986)—place the same kinds of restrictions on chunking strings into constituents as syntactic grammars, including self-embedding of prosodic constituents, and (iv) introducing prosodic constituents and introducing recursion over prosodic constituents into phonological grammar is necessary to capture generalizations and can reduce the computational complexity of the grammar in a precise sense.

An idea that has appeared in the phonological literature is that recursion can only occur in trees and not in strings. For instance, Scheer (2004: xlvi) states: "there is no recursion in phonology because there is no tree-building mechanism in this module" and Scheer (2013: 70) states that: "recursion supposes the existence of trees. It occurs when an item dominates another item of the same kind (e.g. a CP dominates another CP). In an environment without trees, this kind of domination cannot exist." Similarly, Neeleman & van de Koot (2006: 1550) states: "Recursion, projection and long-distance dependencies are characteristic of syntax, but are absent in phonology. This follows if only syntax has trees." Relatedly, it is often assumed that, as stated in the oft-cited definition from Pinker & Jackendoff (2005: 211): "Recursion consists of embedding a constituent in a constituent of the same type, for example a relative clause inside a relative clause…This does not exist in phonological structure: a syllable, for instance, cannot be embedded in another syllable." And there have been proposals about recursion of a broad range of prosodic constituent categories, e.g., from onsets, rimes, and syllables (van der Hulst 2010) to the prosodic word (Selkirk 1996; Peperkamp 1997) to the phonological phrase (Gussenhoven 2005; Schreuder et al. 2009), to the intonational phrase (Ladd 1986).

There is a superfluity of senses in which the term "recursion" is used in mathematical and computational approaches to linguistics. The sense linguists often mean, as evidenced in the quotes above, is the self-embedding of constituents, as exemplified by the constituent of type A in Chomsky & Miller (1963: 290),

redrawn in (1). A constituent is a string chunk that is represented as the sequence of terminal symbols that label the leaves dominated by a single node in a tree. Therefore, if the definition of recursion is taken to be embedding of a constituent in a constituent of the same type, then recursion does presuppose trees as a data structure. But that does not mean that recursion can only occur in trees and not in strings. The trees in (1) are exemplars of *derivation trees*, which record the rewrite rules (e.g., $S \rightarrow NP\ VP$, $A \rightarrow B\ C$) from a generative grammar that are applied to build, i.e., derive, well-formed strings. From this derivational perspective, it is uncontroversial that phonological grammars include trees as data structures as well as strings. And as we explicate in §2, it is also uncontroversial that phonological grammars are recursive, in the sense of constituent embedding in derivation trees.

(1) Types of recursion, redrawn from Chomsky & Miller (1963: Fig. 4, p. 290)



(a) Center-embedding[1] (b) Left-recursive (c) Right-recursive

The difference between syntax and phonology that has been characterized as syntax having trees and phonology having only strings, seen from a derivational lens, is instead a difference in restrictions on the structure of the derivation tree, i.e., restrictions on how the output string can be chunked into constituents, and which string chunks recognized as constituents can further be recognized to have shared properties by being constituents of the same category. These differing restrictions define distinct, well-studied classes of grammars in the Chomsky Hierarchy of grammars (Chomsky & Miller 1963; Chomsky 1956) in formal language theory.

A class of generative grammars that has been claimed to be sufficiently powerful to express all attested properties of phonological strings and transformations in natural language is the class of finite state grammars (FSGs, also known as regular grammars), see Heinz (2011) for an introduction.[2] A finite state grammar derivation tree is limited to self-embedding that is either left-recursive (1b), i.e., on the left edge

1. Chomsky & Miller (1963) refers to this as "self-embedding" but center-embedding is the more common term used in contemporary parlance.
2. Work in computational phonology in the last decade has continued to further restrict the class of grammars need to express phonological patterns to sub-classes of the finite state grammars, see Heinz (2018) for a review.

of the tree, or right-recursive (1c), i.e., on the right edge of the tree, and cannot have both (Chomsky 1963: 394). However, classic rewrite rules like *S* → *NP VP* from introductory syntax lectures belong to the class of context-free grammars and are not finite state. Context-free derivation trees can have left-recursion, and/or right-recursion, as well as center-embedding, i.e., self-embedding internal to the tree (1a). The claim that phonological grammars are finite state is thus a claim that phonological grammars do not have center-embedding, while syntactic grammars can.

In this paper, we restrict our discussion of recursion to the sense of constituent self-embedding in derivation trees, i.e., recursion in grammars. We refer to this sense of recursion as "self-embedding" (s.e.) and use the terms left- and right recursion and center-embedding exemplified in (1) to refer to specific types of self-embedding. Taking the perspective of building up strings and trees, the rest of this paper consists of three main sections. The first, Section 2, explicates how self-embedding is necessary in phonological grammars to build well-formed strings of unbounded length, i.e., strings of arbitrary, finite length. The kinds of grammars we describe in this section, FSGs, have the minimal structural complexity in the Chomsky Hierarchy required to generate strings of unbounded length. These grammars do not have sufficient power to chunk a string the way that prosodic constituents have been proposed to, e.g., they cannot structure *CVCV* as two sister syllable nodes, each picking out a *CV* chunk. While Section 2 begins by motivating self-embedding in phonological grammars by the lack of a principled bound on word length, it ends by showing how the greater motivation for self-embedding is that it is a side effect of recognizing phonological generalizations. The same high-level line of thinking runs through Section 3, which moves from grammars for strings to grammars for trees that can build prosodic constituents. Section 3 also shows how the consideration of how prosodic trees are built up provides a computational perspective that can help elucidate what it means for constituents to be labeled with the same prosodic category and clarify the relation between self-embedding and notions of complexity.

As a final prefacing note, we point out that restricting this paper to the discussion of recursion in grammars means that that we set aside process recursion (see for instance discussions in Parker (2006); Tomalin (2007); Stabler (2014); Idsardi (2018) a.o.), which concerns computing with deferred operations (Abelson et al. 1996, §1.2.1). A grammar defines procedures by which we can derive well-formed strings (or trees). Self-embedding in the definition of the grammar is an instance of procedural recursion. A procedure can be recursive, but be computed with a recursive or an iterative process, see Abelson et al. (1996, §1.2.1) which illustrates this point for the classic example of computing factorials.

2. Self-embedding in building phonological strings

There is no principled upper bound on the length of a word in natural language. To generate all the phonotactically licit words of a language, a phonological grammar must therefore have the capacity to generate words of arbitrary (finite) length. A string rewrite grammar is defined over an alphabet Σ, where an alphabet is a

non-empty, finite set of symbols, e.g., $\Sigma = \{C, V\}$. Rewrite rules in a grammar determine the set of licit strings over the alphabet by restricting which strings can be derived. Derivational steps in finite state string grammars are restricted to two moves: (i) extending a string by a single element in a single direction, e.g., to the right—and only this single direction, over the entire course of the derivation—or (ii) terminating the string. More formally, assuming a non-empty, finite alphabet of terminal symbols, Σ, a finite set of nonterminal categories *Cat* disjoint from the alphabet, and a start symbol category $S \in Cat$ (that initiates the string derivation), a (right-linear) finite state string rewrite grammar is restricted to a finite set of rewrite rules of the form $\alpha \rightarrow a\beta$ and $\alpha \rightarrow \lambda$, where $\alpha, \beta \in Cat$ and $a \in \Sigma$ (Chomsky & Miller 1963) and λ indicates the empty string (the string of length 0). These rewrite rules may look familiar from context-free syntactic rules like $S \rightarrow NP\ VP$, but note that derivations with finite state grammars cannot include $S \rightarrow NP\ VP$ because there are two categories on the right side: *NP* and *VP*.

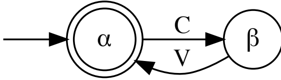
Finite state rules can only add restrictions on string extension steps. For instance, we could define rules to restrict strings to the concatenation of arbitrarily many CV chunks, e.g., *CV*, *CVCV*, *CVCVCVCVCV*. In plain English, the restrictions would be stated as: a string must be initiated with a C, and if we extend with a C, then we must immediately follow this with extending by a V. These restrictions would be stated with the rewrite grammar in (2).

(2) Finite state string rewrite grammar for building strings of arbitrarily many CV chunks
   a. Assume $\Sigma = \{C, V\}$, $Cat = \{\alpha, \beta\}$, start category α
   b. $\alpha \rightarrow C\,\beta$ (rule to extend string with suffix C)
   c. $\beta \rightarrow V\,\alpha$ (rule to extend string with suffix V)
   d. $\alpha \rightarrow \lambda$ (rule to terminate string)

Finite state rewrite grammars are called finite state because they can be modeled with automata with finitely many states and transitions (Sipser 2013, Ch. 1). The grammar in (2) is equivalent to the finite state acceptor (FSA) represented as a transition diagram in (3), an automaton that accepts (or generates) the same set of strings, with the same incremental steps. Both have the same alphabet $\Sigma = \{C, V\}$; the set of categories $Cat = \{\alpha, \beta\}$ is the set of states in the automaton; the start category α is the start state α, indicated by the initial arrow pointing to state α, and the string termination rule $\alpha \rightarrow \lambda$ is equivalent to defining α as a final state, indicated by the double circle. The string extension rules in the re-write grammar correspond to the transition function of the FSA, e.g., $\alpha \rightarrow C\,\beta$ says "if the automaton is in state α and processes (accepts or generates) a C, then it transitions to state β" and $\beta \rightarrow V\,\alpha$ says "if the automaton is in state β and processes (accepts or generates) a V, then it transitions to state α". All and only the strings that can be derived with the grammar in (2) can be generated by (3) by traversing a path from the start state α to the final state α, via allowed transitions between states. Finite state re-write grammars and finite state automatons can thus be thought of as different notations

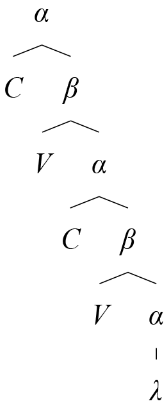
for defining finite state grammars, when we take the broad sense of a grammar as a device that recognizes which string derivations (or tree derivations, for tree grammars) are well-formed.

(3) Transition diagram specifying finite state acceptor (FSA) equivalent to (2) with $\Sigma = \{C, V\}$, states $\{\alpha, \beta\}$, start state α, and final state α



The process of building a string with a generative grammar is standardly represented with a derivation tree. We show the derivation tree for the string *CVCV* with the grammar in (2) in (4). Starting from the start category α at the root of the tree, each successive derivation step, i.e., each rule application, is represented by the expansion of a mother node into its daughter(s), and the derived string can be read off from the leaves in the tree, left to right. Stepping through the derivation tree is equivalent to traversing a corresponding path in the FSA in (3), as shown in (5). We return to the FSA perspective on building up strings to provide insight into self-embedding in §2.2.

(4) Derivation tree for *CVCV* using finite state rewrite grammar (2)



(5) Steps in building *CVCV* string with finite state rewrite grammar in (2) and finite state automaton in (3)

Step	Finite state grammar	Finite state automaton
0	Start with *Cat* α	Start at start state α
1	$\alpha \rightarrow C\,\beta$	Generate C, transition from state α to state β
2	$\beta \rightarrow V\,\alpha$	Generate V, transition from state β to state α
3	$\alpha \rightarrow C\,\beta$	Generate C, transition from state α to state β
4	$\beta \rightarrow V\,\alpha$	Generate V, transition from state β to state α
5	$\alpha \rightarrow \lambda$	End at final state α

Self-embedding is the mechanism that grammars have to build strings of unbounded length. The grammar (2) is right-recursive because of the pair of rules $\alpha \rightarrow C\,\beta$ *and* $\beta \rightarrow V\,\alpha$*:* the category α appears on both the left-hand side of a rule as well as the right-hand side of a rule. The self-embedding in FSG (2) is expressed in corresponding FSA (3) as the cycle between α and β. More generally, self-embedding in an FSG is expressed in an equivalent FSA as a cycle, i.e., a path traversing the FSA that reaches a state that was already previously visited. The derivation tree for *CVCV* in (4) is right-recursive, cf. (1c), because of self-embedding of both α and β on the right edge of the tree. But what are these self-embedded constituents? We described the grammar in (2) as generating strings of arbitrarily many *CV* chunks. From that description, we might expect *CVCV* to be broken down into exactly two *CV*-chunk sister constituents by the grammar: [*CV*][*CV*], where constituents are indicated with brackets.

However, in the derivation tree (4) for *CVCV*, there are four constituents—all the suffixes of *CVCV*: *CVC*[*V*], *CV*[*CV*], *C*[*VCV*], [*CVCV*], with suffix-constituents shown as enclosed in square brackets. (In formal language theory, a suffix (prefix) of string *s* is a substring of *s* that ends (starts) at the end (start) of *s*.) Prefix [*CV*]*CV* is not picked out by the derivation tree as a constituent. This is not an accident: a (right-linear) finite state grammar like (2) can build up a string only by suffixing, resulting in a purely right branching derivation tree (and potentially right-recursion). Or, alternatively a (left-linear) finite state grammar (restricted to prefix extension $\alpha \rightarrow \beta a$ rather than suffix extension $\alpha \rightarrow a\beta$) can build up a string only by prefixing, resulting in a purely left-branching derivation tree and only prefix-constituents (and potentially left-recursion). A finite state grammar cannot chunk a string into both prefix and suffix-constituents, as needed to structure *CVCV* as [*CV*][*CV*]. It also can only pick out string chunks aligned to the edge of the string, and thus cannot chunk out a middle *CV*-unit such as *CV*[*CV*]*CV*.[3]

We can now see how the lack of a principled upper bound on the length of a word implies that there is self-embedding in phonological grammars even if we do not assume the existence of prosodic constituents such as syllables that would provide [*CV*][*CV*] chunking. The discrepancy between the description of the allowed strings generated by the grammar in (2) and the actual string chunks picked out by constituents in the derivation also points to a fundamental issue: *we cannot tell how a string was built only from the information in the string itself*. We elaborate on this point as we consider the implications of boundedness in string length due to the finite realization of unbounded structure in physical systems.

3. Although FSGs defined over $\Sigma = \{C, V\}$ cannot chunk *CVCV* as [*CV*][*CV*], if brackets were admitted into the alphabet so that $\Sigma = \{C, V, [,]\}$, FSGs could define the language $([CV])^n$, i.e., a language with an arbitrary number of repetitions of the string [*CV*]. We discuss bracketed string representations of constituents in §2.3.

2.1. Building up strings of a bounded number of CV chunks

The evidence that there is no principled upper bound on the length of a word obviously does not come from witnessing the actual production of words that are unbounded in length. Rather, it is inferred because the principles that seem to govern word composition up to reasonable lengths do not seem to include size bounds, and the reason we do not see longer examples already has independent explanations coming from finite restrictions on memory, attention, breath, and life. Computers only have finite memory, e.g., the maximum length of an ASCII character string in Python on a 64-bit system is 9223372036854775807.[4] Humans run out of breath and die, so there is no way that a human could utter or sign a word of unbounded length.

The finite realization of physical systems implies that a phonological grammar that only generates strings up to some bounded length would, in practice, be sufficient for modeling observed phonological patterns in natural language: no self-embedding needed, as shown in the non-s.e. finite state grammars that generate strings of up to a bounded number of repetitions of *CV*, which we call "*CV* strings": (6) generates up to one repetition and (7) generates up to two. Corresponding FSAs are given in (8)-(9). The FSAs contain no cycles (they are acyclic) because there is no self-embedding in the corresponding FSGs.

(6) Non-s.e. finite state grammar for building CV-strings of up to one CV chunk

a. Assume $\Sigma = \{C, V\}$, $Cat = \{\alpha, \beta, \gamma\}$, start category α

b. $\alpha \rightarrow C\,\beta$ (rule to extend string with suffix C)

c. $\beta \rightarrow V\,\gamma$ (rule to extend string with suffix V)

d. $\alpha, \gamma \rightarrow \lambda$ (two rules to terminate string)

(7) Non-s.e. finite state grammar for building CV-strings of up to two CV chunks

a. Assume $\Sigma = \{C, V\}$, $Cat = \{\alpha, \beta, \gamma, \delta, \varepsilon\}$, start category α

b. $\alpha \rightarrow C\,\beta$ (rule to extend string with suffix C)

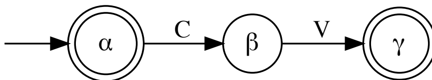
c. $\beta \rightarrow V\,\gamma$ (rule to extend string with suffix V)

d. $\gamma \rightarrow C\,\delta$ (rule to extend string with suffix C)

e. $\delta \rightarrow V\,\varepsilon$ (rule to extend string with suffix V)

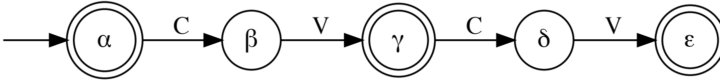
f. $\alpha, \gamma, \varepsilon \rightarrow \lambda$ (three rules to terminate string)

(8) Acyclic FSA for building CV-strings of up to one CV chunk



4. This can be checked with the Python commands `import sys`; `sys.maxsize`.

(9) Acyclic FSA for building CV-strings of up to two CV chunks



In fact, an even less expressive structural class of grammars than non-s.e. finite state grammars is sufficient to generate *CV*-strings up to some bounded string length. (The more possible strings a grammar can generate, the more expressive it is). With our alphabet of size 2, $\Sigma = \{C, V\}$, there are $2^{k+1} - 1$ possible strings for strings up to length k.[5] And in general, the set of possible vowels and consonants in natural language is finitely bounded, so the number of possible strings up to some bounded string length is finite. Therefore, we can simply list out all the licit strings in the grammar. Grammars like this, that simply list the licit strings in a language, form a structural class of grammars that can be called finite grammars, since they can generate only a bounded finite number of words, i.e., a finite language (Nowak et al. 2002). They fail to recognize any generalizations about which strings are licit, so the number of words that can be derived is equivalent to the number of rules in the grammar.

Finite grammars for building up *CV*-strings of up to one and two *CV*-chunks are shown in (10-11). Finite grammars have only a single category, the start category, e.g., α in (10-11). Because finite grammar rules simply list licit strings, $\alpha \rightarrow s$, where s is a string over the alphabet Σ, they only have terminal elements on the right-hand side and do not have the capacity for self-embedding.

(10) Finite grammar for building CV-strings of up to one CV chunk

a. Assume $\Sigma = \{C, V\}$, $Cat = \{\alpha\}$, start category α

b. $\alpha \rightarrow CV$ (rule to generate *CV*)

c. $\alpha \rightarrow \lambda$ (rule to generate empty string)

(11) Finite grammar for building CV-strings of up to two CV chunks

a. Assume $\Sigma = \{C, V\}$, $Cat = \{\alpha\}$, start category α

b. $\alpha \rightarrow CV$ (rule to generate *CV*)

c. $\alpha \rightarrow CVCV$ (rule to generate *CVCV*)

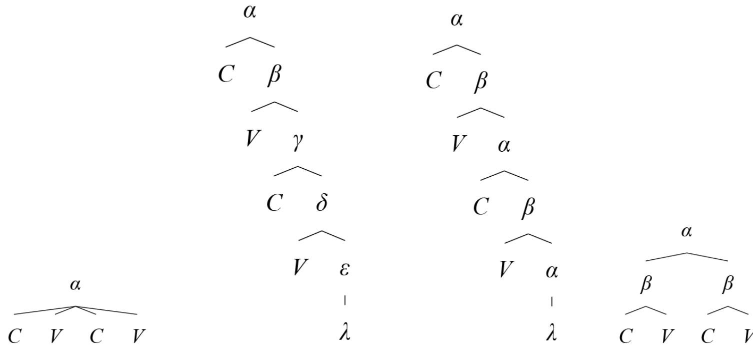
d. $\alpha \rightarrow \lambda$ (rule to generate empty string)

We now have introduced three different kinds of grammars we can use to derive a bounded *CV*-string like *CVCV*. Derivation trees for *CVCV* using these different grammars are shown in (12), plus one that does chunk *CVCV* as two sister constituents (12d). The sample of possible derivation trees for *CVCV* shown—a

5. See Appendix A.

finite grammar derivation using (11), a non-s.e. FSG derivation using (7), a right-recursive FSG derivation using (2), and a context-free grammar (CFG) derivation (explicated in §3.1)—are only four of infinitely many possible derivations of *CVCV*.[6] How do we choose between the different derivation trees and their corresponding grammar types?

(12) Multiple derivations of *CVCV* over the alphabet $\Sigma = \{C, V\}$



(a) Finite grammar (b) Non-s.e. FSG (c) Right-rec. FSG (d) CFG

A standard answer is to compare the relative succinctness of the grammars based on some metric of the "size" of the grammar, e.g., the number of rules, and to choose the most succinct grammar, e.g., see Chomsky & Halle (1968); Meyer & Fischer (1971); Hartmanis (1980). For example, the number of rules needed to generate zero or more repetitions of *(C)V* (where *C* is optional) up to some finite bound of repetitions, *k*, where *k* is a non-negative integer, is $4k + 1$ for a non-s.e. finite state grammar, but $2^{k+1} - 1$ for a finite grammar.[7] Thus, we can say that a non-s.e. finite state grammar for bounded *(C)V*-strings is exponentially more succinct than a finite grammar. But for each additional (*C*)*V* unit allowed, a non-s.e. finite state grammar still needs to add an additional four rules. In comparison, a right-recursive finite state grammar for (*C*)*V*-strings would require only four rules,[8] regardless of the number of (*C*)*V* chunks to be generated—a more succinct grammar than any non-s.e. FSG for some finite bound. The observed data for a language with only (*C*)*V*-strings can only ever be a finite sample, e.g., {*V*, *CV*, *VV*, *CVV*, *VCV*, *CVCV*}, but extending the set of licit strings in the language to be generated beyond that finite sample to the infinite set of (*C*)*V* strings of arbitrary length, {(*C*)*V*}*, allows a reduction in the size of the FSG defined, see Savitch (1993).

6. To see there are infinitely many possible derivations, consider context-free grammars, defined in §3. We could define context-free rules that would allow telescoping out a unary branch from β to the terminal elements *C* and *V* in (12d) with as many layers as we want with derivation steps that rewrite one category as another, e.g., $\beta \rightarrow \gamma$.
7. See Appendix §B for a sketch of a proof.
8. See (28) in Appendix §B.

But what if *C* isn't optional, and we can only build up strings of up to two *CV*-chunks? If we compare the grammars used in the derivations in (12), the finite grammar (11) has three rules; the non-s.e. FSG (7) has seven rules, and the right-recursive FSG (2) has three rules. By our succinctness criterion, the right-recursive FSG is preferable to the non-s.e. FSG because it has fewer rules, but the right-recursive FSG and finite grammar have the same number of rules and thus we can't decide between them. We could also tinker with the operationalization of succinctness and leave rules for terminating strings out of the rule count, in which case the finite grammar has two rules, the non-s.e. FSG has four rules, and the right-recursive FSG has two rules. Regardless of the way we operationalize succinctness, it is clear that when string length is bounded to be as small as a handful of symbols, there is little or no reason to prefer finite or non-s.e. FSGs over s.e. FSGs due to succinctness.

However, even in cases where it takes fewer rules and symbols to write down a finite grammar than a finite state grammar to generate the same set of licit strings, a finite state grammar expresses structural generalizations that a finite grammar doesn't. Compare the non-recursive FSG in (7) that generates $\{\lambda, CV, CVCV\}$ with 7 rules to the finite grammar with just 3 rules (11). The finite grammar recognizes no shared properties between *CV* and *CVCV* other than that they are licit: no string chunks smaller than an entire licit string can be recognized as a constituent, and all licit strings are recognized as constituents of the same category, α. The non-s.e. FSG recognizes at least that the licit strings have a prefix in common: building up *CVCV* begins by building up *CV*, i.e., all (non-empty) words must begin with *CV*. More generally, a non-s.e. FSG can encode phonotactic generalizations aligned to a word edge, e.g., no words in English can begin with an /ŋ/ (for a grammar over a segmental alphabet), all words must end in a vowel in Samoan. Given the prevalence of phonotactic generalizations at word edges in natural language, finite grammars are not a good choice compared to non-recursive FSGs—even in cases where they may be, by some measures, more succinct.

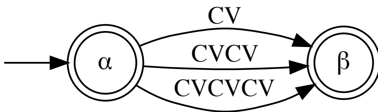
Now comparing the non-s.e. and s.e. FSGs, even if we only consider CV-strings up to two *CV* chunks (with only a tiny numerical difference of 4 vs. 2 string extension rules comparing the non-s.e. to the right-recursive grammar), the two additional rules required in the non-s.e. FSG (7) are copies of the rules in the right-recursive grammar (2), up to category labels. The non-s.e. FSGs fail to notice the generalization that a *C* must always be followed by a *V* and that a (non-empty) string can only end in a *V*. That the licit strings happen to follow this phonotactic restriction is treated as accidental. Thus, even if we only need to be able to build *CV*-strings up to two *CV* chunks, we must do this with the right-recursive grammar to capture the generalization that the only licit strings are built up from concatenating *CV* units. This is reminiscent of the programmer's aphorism: "Two or more, use a for!"

2.2. Self-embedding and generalization

The connection between self-embedding and generalization becomes especially clear when we take the FSA perspective and consider learning. Suppose a language

learner is exposed to *CV*, *CVCV*, and *CVCVCV* (and the empty string) as a sample of learning data, and that only *CV*-strings are licit in the language. If the learner simply memorizes that the strings in the input data comprise the set of licit strings in the language, without noticing any patterns across them, then the learned grammar could be the finite grammar in (11) with an additional rule $\alpha \rightarrow CVCVCV$. We can represent this finite grammar with the FSA in (13) if we define the alphabet to be a list of the three licit non-empty strings, $\Sigma = \{CV, CVCV, CVCVCV\}$, define transitions generating each of those three strings, and force any non-empty string generated by the FSA to terminate after a single transition.[9]

(13) FSA representation of finite grammar that generates exactly the language sample input to the learner



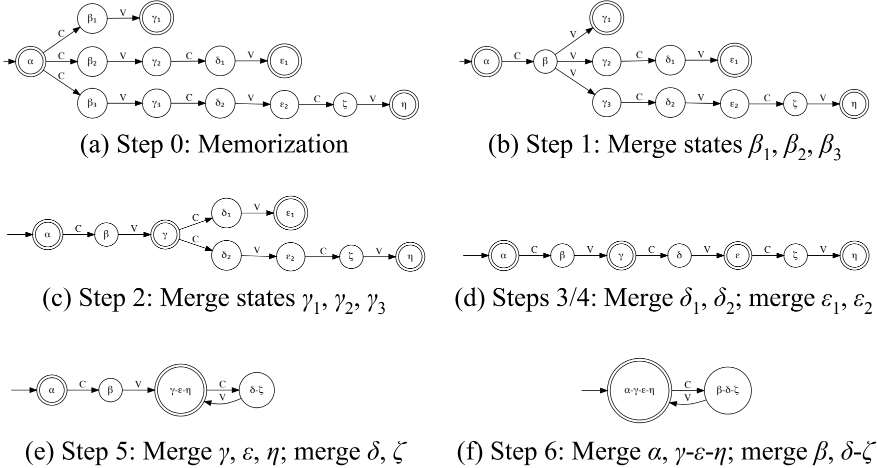
If the FSA in (13) represents the learner's knowledge of licit strings in the language after exposure to an input of *CV*, *CVCV*, and *CVCVCV*, then the learner has faithfully remembered all the distinctions between the licit strings. Generating any of these three strings results in the same, single-transition path through the FSA, from state α to state β. Because the FSA representation of a finite grammar provides no mechanism of decomposing a licit string into its subparts—each licit string is a non-decomposable atom listed in the alphabet—there is no way for such an FSA to recognize any shared properties between the licit strings that were in the input. The learner has no way of noticing that so far, the strings in the input all begin with *CV*, much less to make the inductive leap, "Aha, *all* licit strings begin with *CV*!", because there is no mechanism for the learner to "forget" the rest of a string after the initial *CV*-prefix.

Another possible learned grammar that remembers all the distinctions in the input is the finite state grammar represented by the FSA in (14a). But unlike the FSA in (13), the FSA in (14a) maximally decomposes each licit string into its subparts: each string extension step results in entering a new state. With this unit-by-unit decomposition into individual *C*s and *V*s, the learner leaves open the possibility of recognizing shared properties between licit strings by forgetting certain distinctions between them. First, there is a way for the learner to notice that all the strings in the input so far begin with *CV*, and that both *CVCV* and *CVCVCV* begin with *CVCV*. That is, there are many redundancies in (14a) because *CV* and *CVCV* are both prefixes of *CVCVCV*, so we can compress the FSA by merging states. States β_1, β_2, β_3 have the same set of licit prefixes, $\{C\}$, so they can be merged into a single state β (14b). Similarly, states γ_1, γ_2, γ_3, which have the same

9. The FSA must indicate string termination by having a transition into a final state, so a finite grammar rule like $\alpha \rightarrow CV$ must be rewritten as a finite state grammar rule $\alpha \rightarrow CV\beta$, where β is a final state and there are no transitions out of β.

set of licit prefixes, {*CV*}, can be merged into γ (14c). States δ_1, δ_2, which share the same set of licit prefixes {*CVC*} can also be merged into δ, and states ε_1, ε_2, which share the same set of licit prefixes {*CVCV*}, can be merged into ε (14d).

(14) State merging for generalization in learning



(a) Step 0: Memorization (b) Step 1: Merge states $\beta_1, \beta_2, \beta_3$

(c) Step 2: Merge states $\gamma_1, \gamma_2, \gamma_3$ (d) Steps 3/4: Merge δ_1, δ_2; merge $\varepsilon_1, \varepsilon_2$

(e) Step 5: Merge $\gamma, \varepsilon, \eta$; merge δ, ζ (f) Step 6: Merge α, γ-ε-η; merge β, δ-ζ

The FSA in (14d) has compressed the learning data as much as possible without loss of information by noticing where strings in the language have prefixes in common. But it still does not generalize about what is phonototactically licit in the language beyond the finite learning data sample, nor does it make the inductive leap that the only licit strings are built from repeating *CV*s. The only way to do so is via further state-merging that will result both in self-embedding in the equivalent FSG and in loss of information about certain aspects of the learning data. The decisions about which states will be merged depend on what inductive biases we assume that the learner has.

One kind of bias that has been studied for learning stress patterns is a bias whereby the learner assumes that it is not an accident, if multiple states have the same incoming path of length *k*, e.g., $k = 2$ (see Heinz (2009) and Heinz et al. (2016, §3.5) for introduction). If they do, that is good enough for the learner to treat them as if they were the same and no longer keep track of any distinctions between them further back than 2 steps. With this bias, the learner would merge γ, ε, η, which share the incoming path of length two *CV* and δ, ζ, which share incoming path of length two *VC*, resulting in (14e), where we have labeled the merged states with the original states that were merged, to be explicit. We can compress (14e) into (14f) without any change in the set of licit strings generated. The learner has now made the inductive leap that the only licit strings are built from repeating *CV*s, resulting in the right-recursive phonotactic grammar and cyclic FSA we started with in (2) and (3), in which we revisit the states *α* and then *β* each time we add another *CV*. That is, *self-embedding is a consequence of generalization in learning. Furthermore, the*

unboundedness of string length that is a result of introducing self-embedding into the grammar is simply a side effect of this generalization.

The FSA perspective also provides insight into what a constituent is. Recall that a state in an FSA is equivalent to a category in a finite state grammar. The non-s.e. grammar (7) and corresponding acylic FSA (14d) both build up *CVCV* with the steps shown in the derivation tree in (12b). Prior to the generalization step in (14e), each string extension step in building up *CVCV* with the grammar results in a new string chunk that is not recognized as having any properties in common with any previous string chunk because each node in the derivation tree, i.e., each constituent, is of a different category type. In contrast, the derivation of *CVCV* after the generalization state-merging steps in (14e, f), shown in (12c) (and repeated from (4)), recognizes a shared property between string chunks. The right-recursive grammar partitions the string resulting from each string-extension step in the derivation into two types: α-constituents that result from (rightward) string extension with a *C*, and β-constituents that result from string extension with a *V*. The distinctions lost in the generalization step enable the learner to coarsen the partition of constituents from as fine as possible, to just two categories. This provides a perspective that obscuring or "forgetting" certain distinct properties between different strings to notice shared properties among them is what it means to recognize constituents as the same type. This perspective is different from one commonly brought up in arguments against self-embedding in phonology, e.g., "A constituent is understood to be a particular type of string, and all constituents of a given type exhibit the same properties, regardless of the size or internal structure of the constituent … In addition, by using the same term, Phonological Word, for both types of structures, we obscure the fact that there are, in fact, different types of strings that exhibit distinct properties." (Vogel 2009: 70-71).

2.3. Interim summary and building up to tree grammars

At this point, we have shown that, if we accept that there is no principled bound on the length of a word, then phonological grammars and derivation trees must be self-embedding. We introduced finite state string grammars, their equivalent representations as finite state acceptors, and derivation trees as representations of how strings are built up step by step. We also pointed out, though, that in physical systems such as humans, the realization of an unbounded structure can only ever be bounded. This boundedness implies that a non-s.e. grammar is sufficiently expressive to generate any word observed empirically. In fact, even a finite string grammar that simply lists the licit strings is sufficiently expressive, and for the short string lengths that may be observable from elicited utterances, may even by some measures be more succinct than a finite state grammar. However, only a self-embedding grammar can start to move towards the generalization that a string is built out of repeating chunks such as *CV*-substrings.

We have also disentangled the issues of self-embedding, prosodic constituents, and whether or not phonology is defined over trees rather than or in addition to strings. All self-embedding discussed up to this point has been without reference

to prosodic constituents such as syllables, feet, and prosodic words. And we have shown even the simplest class of grammars—finite grammars that list each licit string—build up strings in a way that is standardly represented as a derivation tree, e.g., the finite grammar derivation tree of *CVCV* in (12a). Finite grammars do not have the capacity for self-embedding. The only constituent is the whole string that was built up: no string chunks smaller than the entire string can be recognized as a constituent. Thus, the finite grammar derivation tree in (12a) underscores that whether a grammar has the capacity for self-embedding is not whether or not it "has trees". Even building up a string in a single step, i.e., simply listing the string, can be represented with the derivation tree as in (12a). Rather, what determines whether or not a grammar has the capacity for self-embedding is what restrictions there are on the derivation tree, which is in turn determined by what restrictions there are on the form of the rewrite rules. A grammar can be self-embedding, so long as the rewrite rules in a grammar are flexible enough to allow rules that have a category on the right-hand side as well as rules that have a category on the left-hand side. Finite state grammars allow such rules, but cannot express the structure of a mother node branching into daughter nodes typical in prosodic trees, e.g., like in a binary foot, Ft → *σσ*.

In moving to building hierarchical prosodic structure, like binary feet, we move from building strings to building trees because the output structure is defined by how it is chunked into subtree constituents. For example, constraints on what kinds of constituents can dominate one another stated in various versions of the Strict Layer Hypothesis, e.g., Selkirk (1996, (4)), are constraints defined on trees, not strings. But all grammars in the Chomsky Hierarchy—finite grammars, finite state grammars and even more expressive grammars such as context free grammars (introduced in §3.1)—are grammars for strings, not trees. They define which strings are acceptable (or can be generated) by the grammar, even if what determines grammaticality is the structure of the derivation tree of the output string.

The rest of the paper extends the derivational and computational perspectives we have taken in discussing string grammars to tree grammars. But first, we note that there is a way to mimic prosodic constituent subtrees with bracketed string representations. For some phonologists, bracketed string representations are just a shorthand notational convenience for trees. For others, e.g., see Neeleman & van de Koot (2006); Idsardi (2018) and references therein, they are crucially strings and not trees, so that phonology is finite state, i.e., so that FSGs are sufficiently expressive to describe all phonological patterns of natural language. Indeed, FSGs may not allow rules like Ft → σσ, but they do allow rules that can generate the string $[_{Ft}[_{\sigma}ma]_{\sigma}[_{\sigma}ma]_{\sigma}]_{Ft}$, assuming the alphabet $\Sigma = \{a, m, [_{Ft}, [_{\sigma},]_{Ft},]_{\sigma}\}$.

The standard mathematical approach one would use to prove that bracketed string representations are in fact insufficient for prosodic structures would be to claim that natural language requires unbounded depth of recursive embedding of prosodic constituents and to notice that bracketed string representations cannot provide that (Dolatian et al. 2021). So long as we only ever need to keep track of a bounded number of layers of recursion over some prosodic category, we can label each layer with distinct symbols, e.g., $[_{\phi_{min}},]_{\phi_{min}}, [_{\phi_{max}},]_{\phi_{max}}$ for self-embed-

ding of ϕ, see also Yu (2019, (31)). But it is impossible to label an unbounded number of layers with distinct symbols, given a finite alphabet. We pointed out in §2.2, though, that unboundedness is really just a side effect of generalization, and arguments from unboundedness don't go through if one is considering the finite realization of physical systems.

There is an even better reason to take prosodic constituents seriously and work with tree representations rather than bracketed strings: bracketed strings make edge effects an accident and can be misleading. As pointed out in Selkirk (1980, §3), since boundary symbols are in the alphabet like any other symbol, it would simply have to be stipulated that they happen to appear only at edges. It would be an accident that strings like $ma]_{\sigma}[_{Ft}[_{\sigma}]_{Ft}m]_{\sigma}[_{\sigma}$ a are ungrammatical, rather than an inherent consequence of the labeled brackets being boundaries. And the fundamental idea of prosodic theory that the edge of a higher prosodic constituent is also the edge of lower prosodic constituents, e.g., the right edge of a foot is also the right edge of a syllable which in turn is also the right edge of a mora, would also be an accident. Without trees, there is nothing for the brackets to structurally be edges of and no inherent ordering due to dominance relations.

As an example of how bracketed string representations can be misleading, even if they can describe the phonological pattern at hand, we briefly consider enclitization patterns in Italian varieties. In Standard Italian, enclitization has no effect on the stress pattern on the lexical word host and enclitics cannot be stressed. In Neapolitan Italian, enclitics can receive stress, with no effect on stress in the lexical word: if there are two enclitics, the first enclitic receives primary stress; if there is only one enclitic, it does not get stressed. In Lucanian Italian, enclitization affects stress on the host: primary stress only occurs on the penultimate syllable of the host plus enclitics, whether that syllable might in the host or a clitic. These patterns are exemplified in (15), based on Peperkamp (1997); Vogel (2009).

(15) Stress patterns in enclitization in varieties of Italian (Vogel 2009, (18)-(19))

Standard Italian		Neapolitan Italian		Lucanian Italian	
véndi	'sell'	cóntə	'tell'	vínnə	'sell'
véndi lo	'sell it'	cóntə lə	'tell it'	vənní llə	'sell it'
véndi me lo	'sell me it'	cóntə tí lə	'tell yourself it'	vinnə mí llə	'sell me it'

Vogel (2009, (20)-(21)) proposes the same prosodic tree for all varieties of Italian: a composite group (CG) with three daughter nodes: (i) a prosodic word (PWd) constituent for the lexical word and (ii) two clitic syllable daughters, and Vogel (2009, (22), (23)) proposes rules for stress assignment for each of the three Italian varieties (16) that operate on this tree. The composite group is taken to be the common domain of stress assignment across the varieties. The difference in stress patterns across the varieties is attributed to the difference in the formulation of the stress assignment rules.

(16) Stress assignment rules defined with labeled bracketed strings (Vogel 2009, (22), (23))

a. Standard Italian: $\sigma_{[+stress]} \rightarrow [+stress]$ $/___ \ldots]_{PW} \ldots]_{CG}$

b. Neapolitan Italian: $\sigma \rightarrow [+stress]$ $/$ $]_{PW}___ \sigma]_{CG}$

c. Lucanian Italian: $\sigma \rightarrow [+stress]$ $/___ \sigma]_{CG}$

The rule for Standard Italian is intended to preserve stress assignment made inside the prosodic word at the level of the composite group (Vogel 2009: 72). There is a mathematically precise way in which we can enforce the idea that Standard Italian grammar cannot dig inside the prosodic word to alter stress assignment once it has already been built: we can define grammars that build trees "bottom-up", constituent by constituent, bottom to top, leaves to root. In such a tree grammar, after the PWd constituent is formed, the grammar cannot peer over the edges of the PWd to dig inside the constituent and affect the syllables within it. Within a bracketed string like (16a), though, if the $[_{PWd}$, $]_{PWd}$ *and* $[_{CG}$, $]_{CG}$ matched bracket pairs are no different from any other pair of terminal symbols in the phonological grammar, then their presence does not imply a barrier in the way that building a PWd constituent in a tree does.

Moreover, in (16), the Lucanian variety seems intuitively the simplest, in the sense that the only boundary symbol that is of relevance in (16c) is $]_{CG}$: stress the syllable that is one syllable to the left of $]_{CG}$.[10] This is misleading, because in the tree, there is no such thing as the syllable that is one over to the left from the right edge of the CG. Rather, there is the syllable that dominates the penultimate leaf, and that syllable could be a daughter of the CG or the PWd node, depending on the number of clitics. Stating the syllable that gets stressed explicitly in terms of the tree structure makes the lack of generalizability about that syllable apparent. This lack of generalizability is also a clear way in which Lucanian Italian in fact has the most complex rather than the simplest stress pattern. One way Lucanian stress can be analyzed is via re-footing inside the prosodic word as clitics are added (Peperkamp 1997: §5.4.2). The next section (§3) introduces finite state bottom-up tree grammars that can compute this kind of "re-structuring" and returns to the Italian example in §3.3.

3. Self-embedding in building prosodic trees

We can extend the FSA perspective introduced for strings in §2 to define grammars for trees. To preview, the FSA perspective reveals that the equivalence between

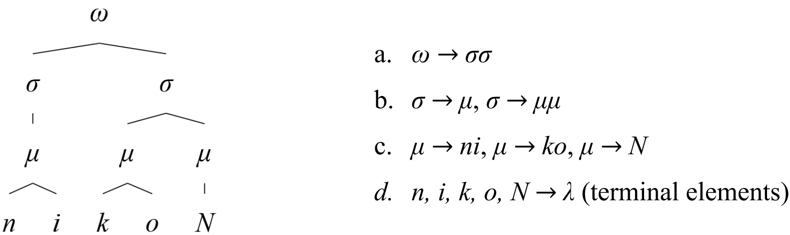
10. In fact, for (16c) to correctly generate all three Lucanian stress assignments from zero to two enclitics, the $]_{PWd}$ symbol must either be erased prior to the application of the stress rule, or ignored, cf. Selkirk (1980: 128). Stated as is, (16c) does not assign any stress in the lexical word or in the enclitic cluster when there are zero or one clitics. The intervening $]_{PWd}$ in $\sigma\ ]_{PWd}\ ]_{CG}$ for zero clitics and $]_{PWd}\ \sigma]_{CG}$ for one clitic prevents the input string from matching the structural description of $\sigma\sigma]_{CG}$. A revision of (16c) that could (accidentally) work would introduce optionality of $]_{PWd}$ in the structural description: $\sigma \rightarrow [+stress] / ___ (\ ]_{PWd}\ )\ \sigma\ (\ ]_{PWd}\ )\ ]_{CG}$.

states in the acceptor and categories in the grammar, see (5), does not hold once we are building up trees rather than strings. The consequences are: (i) a precise mechanism by which prosodic constituents of the same type can still have distinct phonological properties (§3.1), as well as (ii) two kinds of recursion in prosodic trees with differing implications for phonological generalization (§3.2). The FSA tree-building perspective also shows how self-embedding can affect the succinctness and expressiveness of the grammar (§3.3).

3.1. Building prosodic trees: a first example

An example of a typical phonological representation with prosodic constituents is shown in the tree in (17), which shows the Japanese word /nikoN/ 'brand name' chunked into prosodic words (ω), syllables (σ) and moras (μ), taken from Gussenhoven (2018: Table 11.2). The /N/ represents a 'moraic/coda nasal', which can only appear in the second mora of a syllable and is phonologically and phonetically distinct from the /n/ at the beginning of /nikoN/. Moraic nasals assimilate in place to a following stop and surface as uvulars before a pause (Vance 1987). We can read off a possible "top-down" (from roots to leaves) grammar fragment used to build up the string /nikoN/ from the prosodic tree in (17): the alphabet is $\Sigma = \{n, i, k, o, N\}$, the set of categories is $Cat = \{\omega, \sigma, \mu\}$, the start category is ω, and the rules are given in (17).

(17) Prosodic tree for /nikoN/ and corresponding grammar fragment



The ability to chunk the string as [*ni*][*koN*] or [*ni*][*ko*][*N*] (without adding brackets to the alphabet) like in (17) is beyond the right- and left-linear string extension rules of finite state grammars. This is because finite state grammars can only have a single symbol from the alphabet followed (or preceded) by a single category symbol on the right-hand side of a string extension rule (§2). Consequently, each constituent in a finite state derivation smaller than the entire string is nested inside another constituent, e.g., as in [*n*[*i*[*k*[*oN*]]]]. A natural way to allow sisters to be separate, non-nested constituents is to have rewrite rules that have two categories on the right-hand side, such as $\omega \rightarrow \sigma\sigma$, $\sigma \rightarrow \mu\mu$ in (17). One well-studied class of grammars that allow such rules in the derivation tree for the string is the structural class in the Chomsky Hierarchy known as context-free grammars (CFGs). These are restricted to rewrite rules of the form $\alpha \rightarrow (\Sigma \cup Cat)^*$, where $\alpha \in Cat$, i.e., rules that rewrite a category as a (finite) string of symbols drawn from the alphabet and

set of categories. CFGs are the kinds of grammars that are familiar from syntactic derivations using rules like $S \rightarrow NP\ VP$, $VP \rightarrow V$. The derivation trees in (1) are all context-free because they each have a mother node **A** branching into two daughter nodes **B** and **C**.

The prosodic tree in (17) is the derivation tree for the output string *nikoN* and is context-free because it is built with rules with multiple categories on the right-hand side. But, as mentioned in §2.3, what phonologists are interested in is not (only) the string yield *nikoN* from the leaves of the tree, but also the branches of the tree—that is, the derivation tree in (17) itself as the output of the derivation, called the "derived tree".[11] Thus, we want grammars that can build up trees and that recognize which trees are grammatical. In §2.2, we showed how an FSG derivation of a string could be represented with an equivalent sequence of steps through a finite state string acceptor. Similarly, the derivation of a tree like (17) can be represented as a sequence of steps through finite state acceptors—but for trees rather than strings (Baker 1978; Comon et al. 2007). Each step in a string acceptor can grow a string by concatenating in a new substring; each step in a tree acceptor can grow a tree, constituent by constituent, "bottom-up" (from leaves to root) by extending a unary subtree or by merging smaller subtrees to form a bigger subtree.[12]

To work up to self-embedding of prosodic constituents, we first sketch how to build the simple tree in (17) with steps through a bottom-up tree acceptor to explore what it means for prosodic constituents to share the same category label. There are three key points this exercise highlights: (i) in the same way that we cannot tell how a string was built only from the information in the string itself (see (12)), we also cannot tell how a tree was built if all we know is what the tree is, but (ii) unlike in building constituents in FSGs with string acceptors (see (5)), building two prosodic constituents of the same type does not imply also reaching the same state via the same transition in the tree acceptor; thus, (iii) prosodic constituents with the same category label can nevertheless have distinct phonological properties due to distinct derivational histories, as discussed at the end of §2.2.

Recall that well-formed derivations of an finite state string rewrite grammar can be recognized by a string acceptor, e.g., (5), and so finite state string grammars can be defined using the notation of rewrite rules or finite state automata. Well-formed derivations of a finite state tree rewrite grammar can be recognized by a bottom-up tree automaton (Baker 1978; Comon et al. 2007). Here we use the notation of finite state tree automata to define our tree grammars (Rounds 1970). A bottom-up tree automaton can be thought of as a generalization of a (string) finite state automaton that can process multiple branches rather than a single branch (a

11. See Stabler (2019) for a similar perspective on syntax.
12. There are other ways to build up trees, e.g., top-down, see Comon et al. (2007, §1.6, 6.4.2), but we take the bottom-up direction as a starting point since it is a standard way to process a tree and is parallel with processing a string left to right as we have discussed with FSAs. When an FSG (FSA) derives (generates/accepts) a string left to right, as exemplified in the discussion of (4, 5), the categories (or states) can encode some information about what came to the left, a history of the derivation. And going bottom-up in a tree, categories (or states) can encode something about what is beneath them.

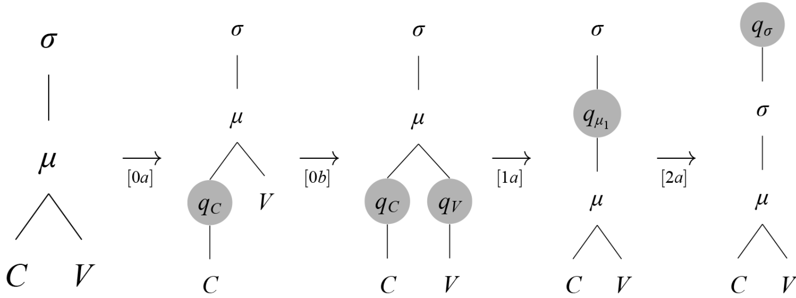
string). A string finite state automaton processes a string from left to right, one symbol at a time, and enters one of finitely many states after each step. A string derivation is recognized as well-formed if and only if the automaton enters a final state after processing the entire string. A bottom-up tree automaton processes a tree from leaves towards the root, one subtree at a time, and enters one of finitely many states after each step. A tree derivation is recognized as well-formed if and only if the automaton enters a final state after processing the tree all the way up to the root.

As a first introduction to tree derivations, we show the grammar and steps to build up just the initial syllable /ni/ in (17) in (18), using the bottom-up tree automaton described by the transition rules in (19).[13] In the rules, the /n/ is subsumed under *C* and /i/ under *V*. The automaton described is a tree transducer, since it not only takes an input tree and processes it like a tree acceptor, but additionally returns an output tree. It is the simplest kind of transducer, an identity transducer, which accepts an input and returns an output identical to the input—a fully faithful mapping from an underlying form to a surface form. The derivation in (18) defines the fully faithful mapping $/\text{ni}/_T \rightarrow [\text{ni}]_T$, where the *T* subscript is a reminder that we are mapping prosodic trees and not just segments. We describe (19) as an identity transducer rather than an acceptor to set the stage for later derivations in the paper where the transductions do change the input tree. All the automata we've been working with up until now have been acceptors and can only compute phonotactic patterns, e.g., licit configurations of sibilants under sibilant harmony, restrictions on voicing word-finally, licit sequences of stressed and unstressed syllables, since they only accept or generate, without modifying the input string or tree. Computing a phonological process, e.g., assimilation, devoicing, stress assignment, tonal insertion, requires a transducer, which specifies an output in addition to an input at each transition.

The rules in (19) take the input tree shown as the leftmost tree in (18) and returns an output tree identical to the input tree as the output tree, the rightmost tree in (18), (ignoring the gray filled circle). A gray filled circle decorating a tree in (18) indicates which state the transducer enters after the application of the transition rule labeling the rewrite arrow to the left of the tree, and the output at each step is shown as the subtree under the state. By convention, a state is positioned as the mother node of the subtree that has just been processed, but isn't actually part of the tree—it's just an annotation like a "you are here" marker. Since the transducer is an identity transducer, (18) shows that the output at each step is simply the subtree that just has been processed. This can be seen in each of the transition rules defined in (19).

13. It might seem strange for us to call (18) a derivation building up the /ni/-syllable tree when the input to the transduction is already the /ni/-syllable tree. But this is no different than when we showed how the finite state string acceptor in (3) derives *CVCV* in (5), taking *CVCV* as an input string. The flip from the generator to the acceptor perspective in (5) can be accomplished by replacing the word "generate" with "accept" and treating the input string as the object incrementally read-in rather than written out. Similarly, to see (18) as a generative process, we treat the input tree as the object incrementally written out rather than read in.

(18) Derivation of /ni/$_T$ → [ni]$_T$ syllable subtree in (17)



(19) Grammar fragment for derivation of /ni/$_T$ → [ni]$_T$; q_σ is a final state

[0*a*]	$C()$	$\rightarrow q_C(C())$	*Enter C leaf*
[0*b*]	$V()$	$\rightarrow q_V(V())$	*Enter V leaf*
[1*a*]	$\mu(q_C(t_1), q_V(t_2))$	$\rightarrow q_{\mu_1}(\mu(t_1, t_2))$	*Merge C, V to build mora*
[2*a*]	$\sigma(q_{\mu_1}(t))$	$\rightarrow q_\sigma(\sigma(t))$	*Build unary syllable*

The left-hand side of a rule shows the structure required for the rule to be applied, and its format differs depending on whether the transducer is at a leaf or a non-terminal node. When the transducer is at a leaf, e.g., Rules [0a], [0b], the left-hand side of the rule is just the leaf, which by definition, has no daughters underneath—indicated by the empty parentheses following the leaf label, e.g., *C*() in Rule [0a]. If the transducer is at a *C* leaf, then Rule [0a] can apply, as shown in the first step in (18). The right-hand side shows the state entered, as well as the output, shown in the immediately following parentheses. For example, Rule [0a] processes the leaf *C*(), transitions the transducer to state q_C, and returns the input leaf *C*() unaltered, as output. And the first step in (18) shows the transducer processing leaf *C* (Rule [0a]) to enter state q_C on the left branch and outputting back *C* on the left branch, which is shown as the daughter of the q_C node. Similarly, the transducer outputs *V* on the right branch after processing leaf *V* (Rule [0b]). Rules [0a] and [0b] could apply in the order shown in (18), or the reverse order, since they process different branches.

The gray circles in (18) move from the leaves towards the root over the course of the derivation since the tree is processed bottom-up. When the transducer is at a non-terminal node, e.g., Rules [1a], [2a], the current node label and the state(s) that the transducer is in must match the left-hand side of a rule for the rule to apply. Rule [1a] is a merge rule that states that if the transducer is at a binary-branching μ node with its left daughter (t_1) in state q_C and its right daughter (t_2) in state q_V, then the transducer can enter q_{μ_1} and output back the μ subtree with its daughters (t_1, t_2) unchanged. The third step in (18) shows the transducer applying this merge rule.

The transduction of the input tree can end successfully if the transducer completes processing the tree up to the root node and enters a final state—a state where the derivation can optionally terminate. Rule [2a] states that if the transducer is

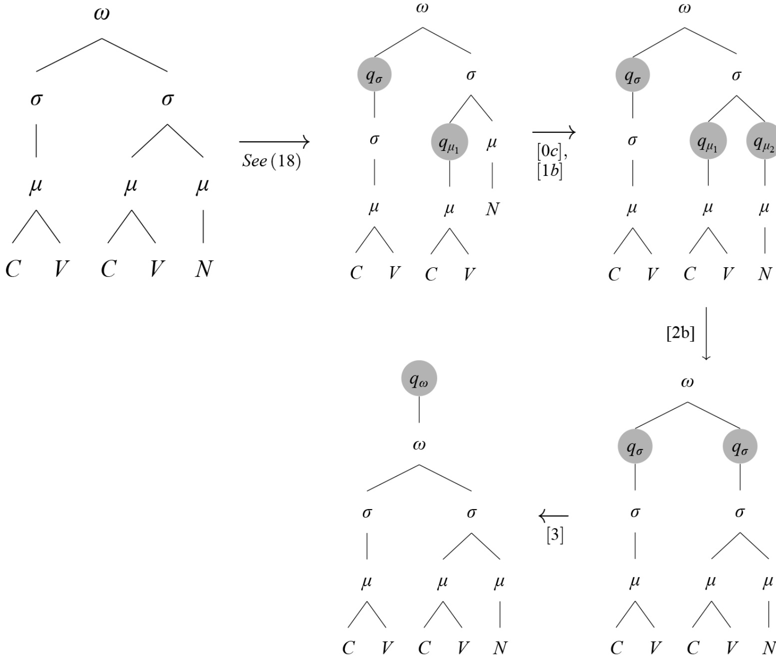
at a σ node with a single daughter (t) in state q_{μ_1}, then the transducer can enter q_σ and output back the σ subtree with its daughter (t) unaltered. For the purposes of processing just a syllable like /ni/ rather than an entire prosodic word like /nikoN/, we designate q_σ as a final state. Upon the application of Rule [2a], the transducer has processed the entire tree up to the root, enters final state q_σ (positioned as the mother node of the root node), and returns the output tree (identical to the input tree), which is shown as the daughter of q_σ. Thus, the tree grammar in (19) recognizes that the derivation of the input tree given in (18) is well-formed and transduces it to an identical output tree.[14]

In fact, that input tree is the only tree that the tree grammar recognizes as well-formed because the steps taken in (18) are the only ones that bring the transducer to a final state upon processing the root. For example, there is no way to build a unary *V*-syllable, since the only rule that builds a μ-subtree requires two daughters, not one. If the input tree to the transducer is a unary *V*-syllable tree, then the transducer does not return an output because that tree is ill-formed according to (19). There is also no way to build a *VC* syllable because there is no way to build a *VC* mora, since Rule [1a] requires a left branch in q_C and right branch in q_V and not vice versa. That a *VC* syllable is ill-formed accurately describes Japanese phonology. But the transducer also does not allow us to build a moraic nasal, i.e., an *N* mora, or a bimoraic syllable, which we need to build the ω-tree for /nikoN/. We need additional rules beyond those in (19) to do that, as shown in (20), using the transition rules stated in (21)—a superset of the rules in (19).

The derivation in (20) for $\text{/nikoN/}_T \rightarrow \text{[nikoN]}_T$ begins with building the left σ-branch to enter q_σ on that branch, following the steps in (18), as well as the application of [0a], [0b], and [1a], just as in (18), to build the second mora and enter q_{μ_1} on that branch. The only difference is that in the grammar in (19), q_σ is not a final state; instead, q_ω is. The first steps not already shown in (18) are to process the moraic nasal leaf *N* with Rule [0c] and then build a μ node on top of that with Rule [1b] to enter q_{μ_2}, and output the processed µ subtree, as shown in the third tree in (20). We show Rules [0c] and [1b] applying in a single step to save space. With the transducer now at a σ node with daughter branches in states q_{μ_1} and q_{μ_2}, Rule [2b] can apply to merge the daughters and build the 2nd σ subtree, and now both daughters of the root node are in state q_σ (4th tree in derivation). Finally, Rule [3] can now apply to merge these two branches, each in state q_σ, to build the ω tree, and the transducer enters final state q_ω upon processing the tree up through the root node and returns an output tree identical to the input tree. This successfully terminates the derivation. Unlike the transducer described in (19), the transducer in (21) can derive more than just one tree. Using the same rules as in (20), we could also derive, for instance, a prosodic word with two *CV* syllables, or one with a *CVN* syllable followed by a *CV* syllable.

14. The definition of a tree acceptor for the /ni/ syllable would differ from the definition of the identity transducer in (19) only by returning no output, i.e., the rules would all contain empty parentheses following the state, e.g., the right-hand side of Rule [1a] would just be q_{μ_1} ().

(20) Derivation trees showing steps for /nikoN/$_T$ → [nikon]$_T$ in (17)



(21) Grammar fragment for building the prosodic tree for /nikoN/$_T$ → [nikoN]$_T$ in (17), q_ω a final state

[0*a*] $C() \rightarrow q_C(C)$ *Enter C leaf*
[0*b*] $V() \rightarrow q_V(V)$ *Enter V leaf*
[0*c*] $N() \rightarrow q_N(N)$ *Enter N leaf*
[1*a*] $\mu(q_C(t_1), q_V(t_2)) \rightarrow q_{\mu_1}(\mu(t_1, t_2))$ *Merge C, V to build σ-initial mora*
[1*b*] $\mu(q_N(t)) \rightarrow q_{\mu_2}(\mu(t))$ *Build unary σ-final mora N*
[1*c*] $\mu(q_V(t)) \rightarrow q_{\mu_1}(\mu(t))$ *Build unary σ-initial mora V*
[2*a*] $\sigma(q_{\mu_1}(t)) \rightarrow q_\sigma(\sigma(t))$ *Build unary syllable*
[2*b*] $\sigma(q_{\mu_1}(t_1), q_{\mu_2}(t_2)) \rightarrow q_\sigma(\sigma(t_1, t_2))$ *Merge μs to build σ*
[3] $\omega(q_\sigma(t_1), q_\sigma(t_2)) \rightarrow q_\omega(\omega(t_1, t_2))$ *Merge σs to build ω*

The sequence of steps in (20) is only one way we could build up the prosodic tree in (17). An alternative is to merge states q_{μ_1} and q_{μ_2} to a single state q_μ, i.e., to replace all instances of q_{μ_1} and q_{μ_2} in (21) with q_μ. Why we didn't choose this alternative becomes evident only when we consider the space of possible trees that could be derived with the rules in (21). While building the tree for /nikoN/ doesn't require Rule [1c], we included it so we could show the consequence of reaching the same state with Rules [1b] and [1c], which both build unary-branching moras.[15]

15. A full grammar for building Japanese prosodic trees would also include a Rule [1d] to build a syllable-final mora containing just *V*, reaching q_{μ_2}.

Having a single state q_μ would fail to distinguish between the initial and final unary-branching moras merged in Rule [2b] to build a bimoraic syllable. We could then build syllables like *NV* even though *N* is licit only as a final mora. Reaching a distinct state upon building a mora with *N* enforces the phonotactic restrictions on syllable shape. State merging like in (14) would overgeneralize over syllable-initial and syllable-final moras: the distinction between them is not one that can be "forgotten".

Another analytic choice that would build a slight variant of the output /nikoN/ tree in (17) would be to have distinct prosodic category types for syllable-initial and syllable-final moras, e.g., μ_1 and μ_2—matching category labels with state labels q_{μ_1} and q_{μ_2} in Rules [1a, 1b, 1c]. This choice would replace the rightmost μ in the tree in (17) with μ_2 and the other two μs with μ_1. Here is where the foundation we lay in §2 with the equivalence between states in finite state string acceptors and FSG categories breaks down once we make the jump to finite state tree acceptors and tree grammar categories. Namely, what labels we give to the categories of the constituents built with Rules [1a, 1b, 1c] has no effect on determining phonotactic restrictions on syllable-initial versus syllable-final moras. Those restrictions arise because of the distinctness of states reached in the derivation, not because of the distinctness of category types in the final output tree. Maintaining distinct category types μ_1 and μ_2 while merging states q_{μ_1} and q_{μ_2} as the single state $q\mu$ would fail to enforce phonotactic restrictions on syllable shape. And so long as we have maintain distinct states q_{μ_1} and q_{μ_2}, the phonotactic restrictions are enforced—regardless of whether there are distinct category types μ_1 and μ_2 or a single category type μ.

What then, is the consequence of two prosodic constituents sharing the same category type? The derivation of the /nikoN/ tree in (17) exemplifies that there are two ways in which nodes in a prosodic tree can end up sharing the same category label: (i) by the choice of the analyst to use common category labels for different constituents built by different rules, like using μ as a common category label across Rules [1a, 1b, 1c], or (ii) as a consequence of applying the same rule, like applying Rule [1a] twice to build the two *CV* moras, which are thus necessarily both of category type μ. From the derivational perspective, when two nodes share the same category label because of the choice of the analyst, it is accidental; when two nodes share the same category label because they were both built by the same rule, it is not. In FSG derivation trees building up strings, the only way for two nodes to share the same category label is because they were built by the same (set of) rule(s). But once we are building trees, the consequence of two prosodic constituents sharing the category type depends on whether it is by the choice of the analyst or due to being built by the same rule. As we just explicated in the case of category type μ shared across Rules [1a, 1b, 1c], when the shared category type is due to the choice of the analyst, it does not result in phonological generalizations based on the category type. When the shared category type is a consequence of the constituents being built by the same rule, though, it can, e.g., if Rule [1a] in (21) is the only rule to build binary-branching moras, then it enforces that all binary-branching moras must be *CV* and not *VV*, or *NC*, etc.

The derived prosodic tree in (17) is non self-embedding and obeys the principles of the Strict Layer Hypothesis (assuming a prosodic hierarchy without a foot), e.g., as given in Nespor & Vogel (1986: 7). Self-embedding of a prosodic category is just a special case of two constituents sharing the same category type, though, so the two different ways constituents can end up sharing the same category label discussed here sets the foundations for our discussion of self-embedding in prosodic trees in the next section. Moreover, a technical but important detail, given the rivalry typically assumed between adhering to the Strict Layer Hypothesis and prosodic self-embedding, is that while the output tree and grammar fragments in (17, 21) are non self-embedding, this is only because any prosodic tree that we write down can only have a finite number of nodes, and so any n-ary branching rule we recover from a tree we write down will be bounded in n, e.g., $\omega \rightarrow \sigma\sigma$, rather than $\omega \rightarrow \sigma\sigma\ldots\sigma$. But in fact, statements of the Strict Layer Hypothesis are clear that there is no principled upper bound on n, e.g., “Following Beckman (1986), branching is held to be n-ary (a node can have any number of daughter nodes)” (Pierrehumbert & Beckman 1988: 21), see also ellipses in Ito & Mester (2003, 1992, (9)). And as discussed in §2, there is no way to generate a sequence of an unbounded number of elements without recursion in the grammar, e.g., $\omega \rightarrow \sigma$, $\omega \rightarrow \sigma + \sigma$, $\omega \rightarrow \sigma + \omega$.

Neeleman & van de Koot (2006, (13)) writes a seemingly non self-embedding rule with a form like $\omega \rightarrow \sigma+$, where $\sigma+$ indicates a sequence of one or more instances of σ. $\sigma+$ is not a string of atomic terminal symbols and category symbols, but a regular expression (see for instance Sipser (2013, §1.3) for an introduction to regular expressions): itself an FSG that generates an unbounded number of strings. Grammars that allow rules like this, with a single category on the left-hand side and a regular expression on the right, have been called extended context-free grammars and are used to abbreviate CFGs, see, e.g., Alberta et al. (2001). Extended CFGs only define languages that CFGs do, but can conceal self-embedding in regular expressions in the right-hand side of a rule. A standard CFG equivalent of the extended CFG rule $\omega \rightarrow \sigma+$ would be the set of rules, with $Cat = \{\omega, \sigma\}$: $\omega \rightarrow \sigma$, $\omega \rightarrow \sigma\sigma$, $\sigma \rightarrow \sigma\sigma$, which makes the recursion explicit. If, alternatively, $\omega \rightarrow \sigma+$ was taken as an infinite list of rules, $\omega \rightarrow \sigma$, $\omega \rightarrow \sigma\sigma$, …, $\omega \rightarrow \sigma\sigma \ldots \sigma$, …, then this list would no longer be a grammar, which is a finite device. Thus, there is a contradiction inherent to the Strict Layer Hypothesis—the generalization afforded by assuming no principled upper bound on n-ary branching is incompatible with a non self-embedding grammar. Without self-embedding, n-ary branching for arbitrary n could only be expressed with an upper bound on n and a set of rules, e.g., $\omega \rightarrow \sigma$, $\omega \rightarrow \sigma\sigma$, $\omega \rightarrow \sigma\sigma\sigma$ for n capped at 3, that would fail to recognize any pattern across the rules.

3.2. Two ways to build self-embedded prosodic constituents

Self-embedding of a constituent in an FSG string derivation tree occurs if and only if the corresponding FSA state labeled with that category type is revisited due to a cycle in the automaton (§2). (There can be no self-embedding of a constituent in the derived string, by definition.) But due to the independence of states and categories

in tree derivations just discussed in §3.1, there are two different sources of self-embedded constituents in a derived tree due to the tree transduction: (i) successive application of a (set of) rule(s) that defines a transition starting and ending in the same state (the analogue of a cycle), and (ii) the application of different rules that happen to process/output a constituent of the same category. Recognizing the distinction between these sources of self-embedding can give us insight into the difference between analyses proposing different projections ("prosodic sub-categories") of the same prosodic category, e.g., minimal and maximal prosodic words, $\omega_{[+min]}$, $\omega_{[+max]}$ (Ito & Mester 2007), and those proposing different categories for each different projection, e.g., *ω* vs. CG (Vogel 2009). The same state on the left and right hand sides of a transition rule (set) implies self-embedding of constituents in the output derived tree, but not vice versa. Moreover, the same state on the left and right hand sides enforces identity between two constituents of the same type built by the same rule(s)—there can be no differences between the phonological processes conditioned by them from that one rule (set).

Exactly such a case has been proposed for Kaqchikel prefixal phonology: "each *ω*-level associated with a high-attaching prefix conditions exactly the same phonotactic patterns" (Bennett 2018: 22). In Kaqchikel, glottal stops are epenthesized word-initially in underlyingly vowel-initial stems to avoid onsetless syllables (Bennett 2018, (5)). This [ʔ]-epenthesis process interacts with prefixation, as exemplified in (22), data from Bennett (2018, (20)). (22a) shows that onsetless (i.e., *V*-initial) monomorphemic stems surface with an initial [ʔ]. (22b) shows that this stem-initial [ʔ]-epenthesis does not occur if the *V*-initial stem is preceded by the prefix /r-/ 'his/her', which becomes an onset. However, stem-initial [ʔ]-epenthesis does occur if the *V*-initial stem is preceded by the prefix /aχ-/ 'agentive', and moreover, the *V*-initial prefix itself surfaces with an epenthetic [ʔ] onset (22c). If the /aχ-/ prefix itself is preceded by the /r-/ prefix, though, it does not surface with an epenthetic [ʔ] onset; instead, the [r-] becomes the onset (22d).

Bennett (2018) analyzes the pattern exemplified in (22) by noticing that there are two classes of prefixes: (i) a "high-attaching" class—which syllabifies separately from the stem and undergoes [ʔ]-epenthesis, including /aχ-/ [ʔaχ-] 'agentive' (Bennett 2018, (10a)) and (ii) a "low-attaching" class, which syllabifies together with the stem, including /r-/ 'his/her' (Bennett 2018, (20)). To account for the distinction between these two classes, Bennett (2018) proposes that low-attaching prefixes merge internal to the minimal prosodic word $\omega_{[+min]}$ (which dominates no other *ω*s) and that syllabification cannot cross *ω* junctures; high-attaching prefixes merge above $\omega_{[+min]}$, and each initiates an additional non-minimal *ω*. Under this analysis, initial [ʔ]-epenthesis occurs only before an onsetless syllable initiating a prosodic word layer. Although Bennett (2018) describes his proposal as "unbounded recursion", stacking of only up to three high-attaching prefixes occurs, e.g. (23)—another kind of finiteness of realization (§2.2), this kind arising from morphosyntax.

(22) [ʔ]-epenthesis and prefixation in Kaqchikel (Bennett 2018, (6))

(a) /ikʼ/	[ʔikʼ]	‘month’	V-initial root
(b) /r-ikʼ/	[r-ikʼ]	‘his/her month’	Low-attaching prefix
(c) /aχ-ikʼ/	[ʔaχ=ʔikʼ]	‘domestic worker’	(AGT-month) High-attaching prefix
(d) /r-aχ-ikʼ/	[r-aχ=ʔikʼ]	‘his/her domestic worker’	High- and low-attaching

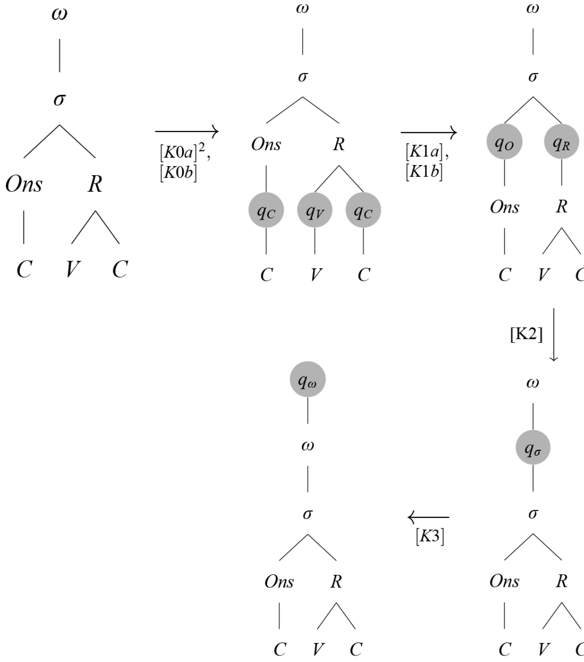
(23) Example of stacking of three high-attaching prefixes in *at achajmak* (Bennett 2018, (22-23))
/at = at͡ʃ = aχ = makʰ/ → [ʔat = ʔat͡ʃ = ʔaχ = makʰ]
‘2SG.ABS = COM = AGT = sin’
‘you are an accomplice’

We illustrate the two ways a derivation can result in self-embedding of a prosodic constituent, using Kaqchikel prefixation as an example. First, as a warm-up, we show the derivation of the prosodic tree transduction for $/\text{r-ik}^{ʔ}/_T$ → $[\text{r-ik}^{ʔ}]_T$ (22b) in (24), which is an identity transduction like (20). Then, we show the derivation for $/\text{ik}^{ʔ}/_T$ → $[\text{ʔik}^{ʔ}]_{T'}$ (22a) in (26): our first example of a non-identity tree transduction, due to the [ʔ]-epenthesis. The subscript change from T to T' in $/\text{ik}^{ʔ}/_T$ → $[\text{ʔik}^{ʔ}]_{T'}$ indicates that the output tree differs from the input tree. Finally, building on that derivation, we show in (27) the derivation of a prosodic tree with self-embedding of the prosodic word for $/\text{at͡ʃ = aχ = mak}^{h}/_T$ → $[\text{ʔat͡ʃ = ʔaχ = mak}^{h}]_{T'}$, the first two high-attaching prefix layers of (23). All three derivations use the transduction rules in the tree grammar fragment in (25).[16] In the grammar, the only final state is q_ω, the state entered when an ω subtree is built. All consonants except [ʔ] are subsumed under C, and all vowels under V.

The tree transduction for $/\text{r-ik}^{ʔ}/_T$ → $[\text{r-ik}^{ʔ}]_T$ is an identity transduction since no [ʔ]-epenthesis occurs. It builds a tree with a ω root node with a single σ-daughter, as shown in the first tree in (24), the input tree, which is identical to the output tree. Since /r-/ is a low-attaching prefix, it is syllabified together with the root inside the same minimal prosodic word.

16. The rules in (25) are only intended to illustrate the aspects of Kaqchikel prefixal phonology touched on in this paper.

(24) Steps for building /r-ikˀ/$_T$ → [r-ikˀ]$_T$ in Kaqchikel following Bennett (2018)



(25) Grammar fragment for building the Kaqchikel prosodic trees in (24, 27) following Bennett (2018), q_ω a final state

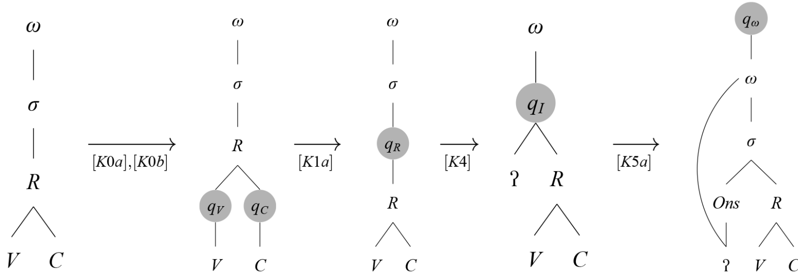
[*K*0*a*]	$C() \rightarrow q_C(C)$		*Enter C leaf*
[*K*0*b*]	$V() \rightarrow q_V(V)$		*Enter V leaf*
[*K*1*a*]	$R(q_V(t_1), q_C(t_2)) \rightarrow q_R(R(t_1, t_2))$		*Merge V, C to build rime*
[*K*1*b*]	$\text{Ons}(q_C(t)) \rightarrow q_O(\text{Ons}(t))$		*Build unary onset*
[*K*2]	$\sigma(q_O(t_1), q_R(t_2)) \rightarrow q_\sigma(\sigma(t_1, t_2))$		*Merge* Ons*, R to build σ*
[*K*3]	$\omega(q_\sigma(t)) \rightarrow q_\omega(\omega(t))$		*Build unary ω*
[*K*4]	$\sigma(q_R(t)) \rightarrow q_I(ʔ, t)$		*Insert ʔ, delete σ*
[*K*5*a*]	$\omega(q_I(t_1, t_2)) \rightarrow q_\omega(\omega(\sigma(\text{Ons}(t_1), t_2)))$		*Attach ʔ, build* Ons*, σ, ω*
[*K*5*b*]	$\omega(q_I(t_1, t_2), q_\omega(t_3)) \rightarrow q_\omega(\omega(\sigma(\text{Ons}(t_1), t_2), t_3)$		*Attach ʔ, build* Ons*, σ, ω*

The derivational steps for /r-ikˀ/$_T$ → [r-ikˀ]$_T$ (24) are very similar to those for /nikoN/$_T$ → [nikoN]$_T$ (20). The first step shown abbreviates three rule applications as a single step: *C* and *V* leaves are processed with [K0a] and [K0b], respectively, the transducer enters q_C and q_V states, and it also outputs the *C* and *V* leaves. The notation [K0a]^2 under the arrow indicates that [K0a] is applied twice. The second step abbreviates the applications of Rules [K1a] and [K1b] as a single step and processes and outputs the *C* onset and *VC* rime, and the transducer enters states q_O and q_R. [K2] then applies to process and output the *σ*-subtree, and the transducer

enters q_σ. Finally, [K3] applies to process and output the ω-tree up to the root node and the transducer enters final state q_ω, successfully terminating the derivation.

The derivation of $/\text{r-ik}^ʔ/_T \rightarrow [\text{r-ik}^ʔ]_T$ we just walked through is still an identity transduction, but the derivation of $/\text{ik}^ʔ/_T \rightarrow [\text{ʔik}^ʔ]_{T'}$ in (26) is our first example of a non-identity transduction. Without the /r-/ prefix attached, /ikˀ/ is onsetless and surfaces with a initial, epenthetic [ʔ]. Since Bennett (2018) describes the [ʔ]-epenthesis as re-syllabification, a natural way to define the mapping $/\text{ik}^ʔ/_T \rightarrow [\text{ʔik}^ʔ]_{T'}$ is to begin with an input tree with a defective syllable—a syllable that has only a rime as a daughter—and to repair the syllable by inserting a [ʔ] onset as a daughter of the syllable node in the output tree. Up through the first two steps shown in (26) that process (and output) the *VC* rime and transition the transducer into state q_R, there's nothing we haven't already seen in the $/\text{r-ik}^ʔ/_T \rightarrow [\text{r-ik}^ʔ]_T$ transduction.

(26) Steps for $/\text{ik}^ʔ/_T \rightarrow [\text{ʔik}^ʔ]_{T'}$ in Kaqchikel following Bennett (2018)



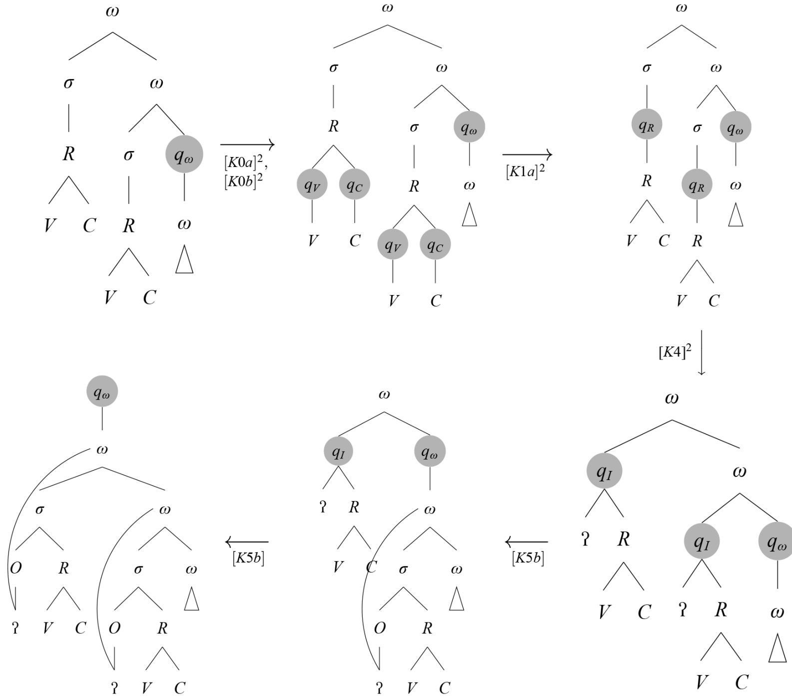
However, the rest of the derivation is new—it first inserts the epenthetic [ʔ] as a sister to the rime and then attaches it as an onset when the ω node is processed. Why insert and attach in two different steps? Attaching [ʔ] as an onset can occur only if the *VC* rime branch in state q_R is ω-initial. Thus, the [ʔ] cannot be inserted as an onset until the tree is processed all the way up through the ω node. Only then does the transducer have information about whether or not the rime is ω-initial. At the same time, the transducer also can't build the σ with just R in it and then insert the onset [ʔ] later, since bottom-up tree transductions cannot change structures that have already been created. Rather, the onset, syllable, and prosodic word nodes need to be processed all at once so that the transducer can "wait" until the ω node is being processed to insert [ʔ] as an onset. That ω formation is precisely what forces the last-resort [ʔ]-epenthesis to avoid an onsetless syllable, due to syllabification being ω-bounded.

For all those nodes to be processed at once and [ʔ]-insertion to be dependent on the ω node, we need to introduce a new kind of transduction step: a "multi" step which requires carrying multiple subtrees up the derivation rather than just one (Lilin 1978; Engelfriet et al. 2009). [K4] and [K5a], as well as [K5b] (not used in (26)) are "multi" steps. Unlike any of the other transition rules, they include a state followed by more than one tree in parentheses, e.g., $q_I(t_1, t_2)$. As shown in (26), [K4] processes the σ node, inserts [ʔ] as a sister to the rime, deletes the σ node, and

takes the transducer to state q_I. Unlike any other tree resulting from a transduction step so far, the result of Rule [K4] has a q_I gray circle that is not positioned as a mother node to a constituent because [ʔ] and *R* have not been merged to build a constituent. The deletion of the σ node allows the [ʔ] and the rime subtree to be visible to the transduction when the ω node is processed. From q_I, Rule [K5a] processes the root ω nodes and inserts Ons and σ all at once to build an onset with daughter [ʔ], merge the onset with the *VC* rime to build a σ-constituent, build an ω-constituent with the σ-constituent as its daughter, successfully terminate in the final state q_ω, and output the output tree. As is often done in similar syntactic derivations, we can interpret the output from Rule [K5a] as a multidominance structure (Gärtner 2002), where a single terminal node has two parents. The final tree explicitly indicates the multiple dependencies of the [ʔ] by showing it as having two mothers: (i) the Ons node, at its original insertion site as sister to the rime, and (ii) the ω node, the node processed when it is attached.

We need only the addition of a variant of Rule [K5a], which we call Rule [K5b], to move from deriving /ikˀ/$_T$ → [ʔikˀ]$_{T'}$ in (26) to deriving /at͡ʃ = aχ = makʰ/$_T$ → [ʔat͡ʃ = ʔaχ = makʰ]$_{T'}$, shown in (27). This transduction is our first that builds self-embedded prosodic constituents, that is, three ω-layers. The transduction begins with the steps in (24) to process the minimal prosodic word, $\omega_{[+min]}$, for the stem /makʰ/$_T$ → [makʰ]$_T$ (not shown). The stem already has an onset, so no [ʔ]-epenthesis occurs at the left edge of $\omega_{[+min]}$. At this point, the transducer is in final state q_ω and can optionally terminate. This is the starting point for the input tree shown in (27). If the derivation were only of /makʰ/$_T$ → [makʰ]$_{T'}$, we could successfully terminate here. But the derivation continues because there are two prefixes to merge in and the transducer is not yet at the root node. Both prefixes are *V*-initial and high-attaching and thus initiate ω layers and force [ʔ]-epenthesis ω-initially. (27) shows that the two [ʔ]-epentheses take almost the same steps as those already shown in /ikˀ/$_T$ → [ʔikˀ]$_{T'}$ in (26), i.e., Rules [K4, K5b]. Rule [K5b] differs from [K5a] (used in (26)) only in processing an ω node that has an ω daughter in addition to a branch in state q_I.

(27) Transduction of /at͡ʃ = aχ = makʰ/$_T$ → [ʔat͡ʃ = ʔaχ = makʰ]$_{T'}$ in Kaqchikel following Bennett (2018)



Holding component parts separately in memory and carrying them up the derivation until they are ready to be assembled, like in Rules [K4, K5a, K5b], pushes up the expressivity of the grammar. In general, any kind of process described as re-syllabification, re-footing, stress shift, etc., in phonology (and, for instance, movement or binding in syntax) requires "multi" rules that carry forward more than one tree up the derivation. This is because processes described as restructuring require defining relations between two or more locations in the prosodic tree. In Bennett (2018)'s proposal of ω-initial [ʔ]-epenthesis, the glottal stop enters as a leaf (location one) since it's a segment but then it is merged to build an onset only when it becomes the leftmost daughter of a ω-node being built (location two). Rules [K4, 5a, 5b] push the tree grammar in (25) into the class of multi bottom-up tree transducers. String yields from trees that can be built with finite state bottom-up tree transducers are context-free, i.e., strings that can be derived with CFG grammars (Comon et al. 2007, §2.4). String yields from trees that can be built with multi finite state bottom-up tree transducers are strings that can be derived with multiple CFGs (Engelfriet et al. 2009), grammars that that are more expressive than CFGs, in which one constituent can enter into relationships with two of its ancestors, e.g., in syntactic movement, see Clark (2014) for an introduction.

Rule [K5b] expresses the generalization, that once we have already built a ω, we are in a state (q_{ω}) where we can merge in a prefix, which surfaces with an initial [ʔ] if the prefix is *V*-initial. And once we have merged in a prefix, we are still in the same state where we can merge in another prefix, with the same phonotactically-driven process of [ʔ]-epenthesis. Every time Rule [K5b] is successively applied, it builds another ω because the output tree from applying it is labeled as an ω. Self-embedding of ω is a consequence of generalization provided by Rule [K5b], cf. discussion in §2.2. There is no way to build a constituent of a distinct category with each successive prefixation, except by copying Rule [K5b] into a set of rules differing only in the category label for the constituent formed and state reached. It would then be an accident that these rules are copies up to category and state labels—a missed generalization and a missed opportunity for succinctness in the grammar (recall, "two or more, use a for!"), even if only up to three prefixes can be stacked. The unboundedness introduced by Rule [K5b] is merely a side effect of encoding the phonological generalization of a process conditioned by prosodic structure.

The comparison of Rule [K5b] with Rule [K3]—both of which build ωs, also illustrates the difference between the two kinds of self-embedding possible in a derived prosodic tree. The self-embedding of ω due to building a constituent with Rule [K3] and then with Rule [K5b] is the kind of recursion of a prosodic category that is "accidental" due to the choice of the analyst. We could have labeled the output tree from Rule [K5b] as a composite group or clitic group, in which case the application of Rule [K3] followed by the application of Rule [K5b] would not have resulted in self-embedding and successive applications of Rule [K5b] would have resulted in self-embedding of clitic groups rather than prosodic words. Bennett (2018, fn. 6) dismisses this possibility, but the fact that $\omega_{[+min]}$ is built by one rule (Rule [K3]), while all $\omega_{[-min]}$s are built by another (Rule [K5b]) provides a mechanism by which an $\omega_{[+min]}$ could exhibit different phonological behavior from all other ω projections, or by which the constituent built by Rule [K3] and those built by Rule [K5b] could be distinct categories. From the point of view of the transduction building the prosodic tree in (27), whether Rule [K3] and Rule [K5b] build constituents of the same type or not is inconsequential. The consequence of choosing between allowing self-embedded prosodic constituents of some category or giving those constituents distinct category labels thus depends on the source of the self-embedding in the derivation: (i) multiple application of a rule that begins and ends in the same state that enforce building constituents of the same category, like Rule [K5b] being applied twice, or (ii) the application of multiple rules which happen to build prosodic constituents of the same category, like Rules [K3] and [K5b].

3.3. Prosodic constituency, self-embedding, and generalizability

Having introduced how the multiple dependencies in "re-structuring" analyses can be modeled with "multi" steps in bottom-up tree transductions, we can return to the interaction of stress assignment and enclitization across varieties of Italian (15)

from §2.3 for one final brief case study focusing on generalizability and prosodic tree building.[17] While additional details of Italian phonology are relevant for a full analysis of stress assignment in Italian, our goal here is not to argue for a particular analysis, but rather to show how a computational perspective from tree transduction can help illuminate the consequences of aspects of different analyses.

With respect to stress assignment, adding one clitic is the same as adding yet another in Standard Italian: the process of merging in a clitic is generalizable. No "multi" steps are necessary as clitics are stacked and enclitization could be expressed in a single transduction rule with the same state on the left- and right-hand sides that merges a syllable with a PWd or CG subtree (depending on the analysis chosen) to build a new PWd/CG, adding a layer for each clitic, much like in Kaqchikel prefixation. Introducing this kind of transduction rule for encliticization (not an analysis proposed by Vogel or Peperkamp, but nevertheless a possible one) would be an alternative to: (i) having separate, unrelated tree transduction rules for each different number of clitics that can be merged at once (see, e.g., end of §3.1), or (ii) introducing "multi" steps to hold clitics until the final clitic is added to then merge all the clitics at once at a single node. Relative to the two other alternatives mentioned, introducing a rule with the same state on the left- and right-hand sides could affect the complexity of the chosen analysis by: (i) increasing the succinctness of the grammar by reducing the number of rules needed to add in clitics to just one, and (ii) limiting the expressivity of the grammar class to standard rather than multi tree transducers. No such rule for adding a clitic would be possible for Neapolitan or Lucanian because there is no generalization to be made: adding one clitic is not the same as adding yet another because stress assignment changes if one clitic vs. two clitics are added in these two varieties. In fact, a natural way of implementing Peperkamp (1997, §5.4.2)'s Lucanian analysis of re-footing inside the prosodic word as clitics are added would require a "multi" step carrying up even syllables in the lexical word until the final clitic is added and stress assignment is determined. However, there is a way to make a generalization about the interaction between clitics and stress in Neapolitan Italian: by connecting stress assignment with building a foot.

Peperkamp's analyses propose that syllables are footed within the prosodic word and that differing stress patterns across Italian varieties reflect differences in the derived prosodic tree (Peperkamp 1997, §5.4)). And in the derived tree for Neapolitan Italian, the host "inner" prosodic word (lexical word) and the clitic foot are daughters of an "outer" prosodic word (if there is only one clitic, then its syllable is a daughter node of the "outer" prosodic word), resulting in a recursive PWd category (Peperkamp 1997, §5.4.1)). Adding one vs. two enclitics changes stress patterns because feet are minimally binary for the light syllables of the enclitics, just like for non-clitic syllables. The generalization that merging two syllables builds a foot and assigns stress on the first syllable affects the succinctness or expressivity of the grammar even if there is an upper bound of two enclitics. Without the introduction of stress assignment as a consequence of building a foot, Neapolitan Italian would require either separate rules for

17. We leave the specification of grammar fragments as an exercise for the reader.

merging different number of enclitics, or a "multi" rule to carry up the first enclitic until the second is added. Finishing stress assignment as two syllables are merged to build a trochaic foot instead allows two clitics to be merged in together as a single stressed foot subtree with a PWd subtree. And under a natural interpretation of Peperkamp (1997: §5.4.1)'s analysis, building a minimal PWd (with no enclitics) or a PWd with a clitic foot reaches a final state q_ω from which yet another clitic foot could be merged to build another PWd layer. That is, like in the *CV*-string state-merging example in §2.2 or the introduction of Rule [K5b] in Kaqchikel prefixation in §3.2, the presence of a transition rule with the same state on the left- and right-hand sides, as well as unboundedness, are consequences of recognizing a phonological generalization.

4. Conclusion

The tree transduction perspective on stress assignment in Neapolitan Italian shows one way a phonological grammar can encode the generalization that "word stress is a constituent (a foot)" (Gussenhoven 2018: 389): via a derivational step that merges two syllables to build a foot and stresses a syllable (indicated with str()), e.g., $\mathrm{Ft}(q_\sigma(t_1), q_\sigma(t_2)) \rightarrow q_{Ft}(Ft(str(t_1), t_2))$. This tree grammar transduction rule is a natural way to directly define stress assignment as part of the process of building a foot: "Simply put, if the representations are right, then the rules will follow" (McCarthy 1988: 43). This connection between stress assignment and prosodic structure could be imitated with grammars for strings and bracketed strings, but only "accidentally", in the sense described in §2.3. While the examples in this paper have generally involved prosodic domains at the level of the prosodic word and below, above the prosodic word, too, bracketed strings representations mimicking subtree constituents miss generalizations. If prosodic trees are transduced from syntactic trees, e.g., as in Match Theory (Selkirk 2011), and constituency information is passed from syntax to phonology via strings marked up with brackets, e.g., see Idsardi (2018: 215-16), any phonologically-conditioned changes to constituency passed in from syntax, too, would be accidental.

Computational perspectives from string grammars have provided substantial insights about phonological patterns, but perspectives from tree grammars have much to offer as well. The insights explicated here illuminating perennial debates about recursion and constituency in phonology just scratch the surface.

References

Abelson, Harold, Gerald Jay Sussman & Julie Sussman. 1996. *Structure and interpretation of computer programs*, 2nd edn. Cambridge, MA: MIT Press.

Alberta, Jürgen, Dora Giammarresi & Derick Wood. 2001. Normal form algorithms for extended context-free grammars. *Theoretical Computer Science* 267: 35-47. <https://doi.org/10.1016/S0304-3975(00)00294-2>

Baker, Brenda. 1978. Tree transducers and tree languages. *Information and control* 37: 241-266.

Bennett, Ryan. 2018. Recursive prosodic words in Kaqchikel (Mayan). *Glossa* 3(1): 67-133.
<https://doi.org/10.5334/gjgl.550>

Chomsky, Noam. 1956. Three descriptions of language. *IRE Transactions in Information Theory* 2(3): 113-124.

Chomsky, Noam. 1963. Formal properties of grammars. In R. Duncan Luce, Robert B. Bush & Eugene Galanter (eds.). *Handbook of mathematical psychology*, Vol. 2, 323-418. New York and London: John Wiley & Sons, Inc. Chap. 12.

Chomsky, Noam & Morris Halle. 1968. *The sound pattern of English*. Cambridge: The MIT Press.

Chomsky, Noam & George A. Miller. 1963. Introduction to the formal analysis of natural languages. In R. Duncan Luce, Robert B. Bush & Eugene Galanter (eds.). *Handbook of mathematical psychology*, Vol. 2, 269-321. New York and London: John Wiley & Sons, Inc. Chap. 11.

Clark, Alexander. 2014. An introduction to multiple context-free grammars for linguists. Retrieved from <https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.714.8708>.

Comon, H., M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison & M. Tommasi. 2007. Tree Automata Techniques and Applications. Retrieved from <http://www.grappa.univ-lille3.fr/tata>. Release October, 12th 2007.

Dolatian, Hossep, Aniello De Santo & Thomas Graf. 2021. Recursive prosody is not finite-state. In *Proceedings of SIGMORPHON 2021*.

Engelfriet, Joost, Eric Lilin & Andreas Maletti. 2009. Extended multi bottom-up tree transducers: Composition and decomposition. *Acta Informatica* 46(8): 561-590.
<https://doi.org/10.1007/s00236-009-0105-8>

Gärtner, Hans-Martin. 2002. *Generalized transformations and beyond*. Berlin: Akademie Verlag.

Gussenhoven, Carlos. 2005. Procliticized phonological phrases in English: Evidence from rhythm. *Studia Linguistica* 59: 174-193.

Gussenhoven, Carlos. 2018. Prosodic typology meets phonological representations. In Larry M. Hyman and Frans Plank (eds.). *Phonological typology*, 389-418. Berlin/Boston: Walter de Gruyter GmbH.

Hartmanis, Juris. 1980. On the succinctness of different representations of languages. *SIAM Journal on Computing* 9: 114-120.

Heinz, Jeffrey. 2009. On the role of locality in learning stress patterns. *Phonology* 26(02): 303-351.
<https://doi.org/10.1017/S0952675709990145>

Heinz, Jeffrey. 2011. Computational phonology – part I: Foundations. *Language and Linguistics Compass* 5(4): 140-152.
<https://doi.org/10.1111/j.1749-818X.2011.00269.x>

Heinz, Jeffrey. 2018. The computational nature of phonological generalizations. In Larry M. Hyman and Frans Plank (eds.). *Phonological typology*, 126-195. Berlin/Boston: De Gruyter Mouton.

Heinz, Jeffrey, Colin de la Higuera & Menno van Zaanen. 2016. *Grammatical inference for computational linguistics*. California: Morgan and Claypool Publishers.

Idsardi, William J. 2018. Why is phonology different? No recursion. In Ángel J. Gallego & Roger Martin (eds.). *Language, Syntax, and the Natural Sciences*, 212-223. Cambridge: Cambridge University Press. <https://doi.org/10.1017/9781316591529.012>

Ito, Junko & Armin Mester. 1992. Weak layering and word binarity. Manuscript. University of California Santa Cruz.

Ito, Junko & Armin Mester. 2003. Weak layering and word binarity. In Takeru Honma, Masao Okazaki, Toshiyuki Tabata & Shin'ichi Tanaka (eds.). *A new century of phonology and phonological theory: a festschrift for Professor Shosuke Haraguchi on the occasion of his sixtieth birthday*, Vol. Revised version of Report LRC-92-09, Linguistic Research Center, University of California, Santa Cruz (1992), 26-65. Tokyo, Japan: Kaitakusha.

Ito, Junko & Armin Mester. 2007. Prosodic adjunction in Japanese compounds. *Formal Approaches to Japanese Linguistics: Proceedings of FAJL 4* 55: 97-111.

Ladd, D. Robert. 1986. Intonational phrasing: The case for recursive prosodic structure. *Phonology Yearbook* 3: 311-340.

Lilin, Eric. 1978. Une generalisation des transducteurs d'etats finis d'arbres: les S-transducteurs. Thése 3éme cycle, Université de Lille.

McCarthy, John J. 1988. Feature geometry and dependency: a review. *Phonetica* 43: 84-108.

Meyer, A. R. & M. J. Fischer. 1971. Economy of description by automata, grammars, and formal systems. In *12th Annual IEEE Symposium on Switching and Automata Theory*, 188-191.

Neeleman, Ad & J. van de Koot. 2006. On syntactic and phonological representations. *Lingua* 116: 1524-1552. <https://doi.org/10.1016/j.lingua.2005.08.006>

Nespor, Marina & Irene Vogel. 1986. *Prosodic phonology*. Dordrecht, The Netherlands: Foris Publications.

Nowak, Martin A., Natalia L. Komarova & Partha Niyogi. 2002. Computational and evolutionary aspects of language. *Nature* 417(6889): 611-617. <http://dx.doi.org/10.1038/nature00771>

Parker, Anna R. 2006. *Evolution as a constraint on theories of syntax: the case against Minimalism*. PhD diss, University of Edinburgh, Edinburgh, Scotland.

Peperkamp, Sharon Andrea. 1997. *Prosodic words*. PhD diss, Universiteit van Amsterdam, Amsterdam.

Pierrehumbert, Janet & Mary Beckman. 1988. *Japanese tone structure*. Cambridge: The MIT Press.

Pinker, Steven & Ray Jackendoff. 2005. The faculty of language: what's special about it? *Cognition* 95: 201-236.

Rounds, William C. 1970. Mappings and grammars on trees. *Mathematical Systems Theory* 4(3): 257-287.

Savitch, Walter J. 1993. Why it might pay to assume that languages are infinite. *Annals of Mathematics and Artificial Intelligence* 8: 17-25.

Scheer, Tobias. 2004. *A lateral theory of phonology*. Berlin/New York: Mouton de Gruyter.

Scheer, Tobias. 2013. Why phonology is flat: the role of concatenation and linearity. *Language Sciences* 39: 54-74. <https://doi.org/10.1016/j.langsci.2013.02.004>

Schreuder, Maartje, Dicky Gilbers & Hugo Quené. 2009. Recursion in phonology. *Lingua* 119: 1243-1252.
<https://doi.org/10.1016/j.lingua.2009.02.007>

Selkirk, Elisabeth. 1980. Prosodic domains in phonology: Sanskrit revisited. In M. Aronoff and M. L. Keans (eds.). *Juncture*. Saratoga, California: Anma Libri.

Selkirk, Elisabeth. 1996. The prosodic structure of function words. In James Morgan & Katherine Demuth (eds.). *Signal to syntax: Bootstrapping from speech to grammar in early acquisition*, 187-213. Mahwah: Lawrence Erlbaum Associates.

Selkirk, Elisabeth. 2011. The syntax-phonology interface. In John Goldsmith, Jason Riggle & Alan C. L. Yu (eds.). *The handbook of phonological theory*, 435-484. Malden: Wiley-Blackwell.
<https://doi.org/10.1002/9781444343069.ch14>

Selkirk, Elisabeth O. 1984. *Phonology and syntax: the relationship between sound and structure*. Cambridge, MA: MIT Press.

Sipser, Michael. 2013. *Introduction to the theory of computation*, 3rd edn. Boston, MA: Cengage Learning.

Stabler, Edward P. 2014. Recursion in grammar and performance. In Tom Roeper & Margaret Speas (eds.). *Recursion: complexity in cognition*, 159-177. Cham: Springer.

Stabler, Edward P. 2019. Three mathematical foundations for syntax. *Annual Review of Linguistics* 5(1): 243-260.
<https://doi.org/10.1146/annurev-linguistics-011415-040658>

Tomalin, Marcus. 2007. Reconsidering recursion in syntactic theory. *Lingua* 117: 1784-1800.
<https://doi.org/10.1016/j.lingua.2006.11.001>

van der Hulst, Harry. 2010. A note on recursion in phonology. In Harry van der Hulst (ed.). *Recursion and human language*, 301-341. Berlin/New York: De Gruyter Mouton. Chap. 17.

Vance, Timothy J. 1987. *An introduction to Japanese phonology*. Albany, NY: State University of New York Press.

Vogel, Irene. 2009. Universals of prosodic structure. In Sergio Scalise, Elisabetta Magni & Antonietta Bisetto (eds.). *Universals of language today*, 59-82. Cham: Springer.

Yu, Kristine M. 2019. Parsing with Minimalist Grammars and prosodic trees. In Robert C. Berwick & Edward P. Stabler (eds.). *Minimalist parsing*, 69-109. Oxford, UK: Oxford University Press.

A. Sketch of proof for number of strings that can be built from Σ = {*C, V*}

There is only one string of length 0: λ. Extending a string built over an alphabet $\Sigma = \{C, V\}$ by one symbol presents two options: concatenating in a *C* or concatenating in a *V*. Thus, each symbol in such a string represents a "slot" with two choices; a string of length 2 has two slots and thus $2 \times 2 = 4$ possibilities; a string of length 3 has $2^3 = 8$ possibilities. The total number of possible strings for strings up to length 3 is therefore $2^0 + 2^1 + 2^2 + 2^3$. In general, the number of possible strings that can be generated over an alphabet of size 2 for strings up to length *k,* where *k* is a non-negative integer, is $\sum_{n=0}^{k} 2^n = 2^{k+1} - 1$.

B. Sketch of proof for succinctness comparison between grammars for generating *C*(*V*) repetitions

Both a non self-embedding FSG and a finite grammar will need one rule to generate an empty string, e.g., $Cat_0 \rightarrow \lambda$.

For a non self-embedding FSG, to generate up to *n* (*C*)*V* repetitions for $n = 1$ requires 4 other rules: $Cat_{n-1} \rightarrow C\ Cat_n$, $Cat_{n-1} \rightarrow V\ Cat_{n+1}$, $Cat_n \rightarrow V\ Cat_{n+1}$, $Cat_{n+1} \rightarrow \lambda$. Without self-embedding, generating from 1 to $n = k$ (*C*)*V* repetitions, $k \geq 1$, requires *k* copies of each of the 4 rules needed for $k = 1$, plus one rule for the empty string. Therefore, in general, generating up to $n = k$ (*C*)*V* requires $4k$ rules, plus one for the empty string, so $4k + 1$ in total.

For a finite grammar, each possible non-empty string requires an additional rule beyond the empty string rule. For each (*C*)*V* repetition, there are 2 possible substrings: *C* and *CV*, so there are 2×2 possible (*C*)*V*(*C*)*V* strings, $2 \times 2 \times 2$ possible (*C*)*V*(*C*)*V*(*C*)*V* strings, and so on. Thus, there are $1 + 2 + 4$ possible strings for 0 to *k* (*C*)*V* repetitions, where $k = 2$, and in general, $\sum_{n=0}^{k} 2^n = 2^{k+1} - 1$ possible strings for any *k*, see (A) for the same kind of argument.

In comparison, with a self-embedding FSG, four rules suffice to generate an unbounded number of (*C*)*V* repetitions (and nothing else):

(28) Finite state grammar for building strings of arbitrarily many (*C*)*V* chunks
  a. Assume $\Sigma = \{C, V\}$, $Cat = \{\alpha, \beta\}$, start category α
  b. $\alpha \rightarrow V\,\alpha$
  c. $\alpha \rightarrow C\,\beta$
  d. $\beta \rightarrow V\,\alpha$
  e. $\alpha \rightarrow \lambda$