Recursive prosody is not finite-state

Hossep Dolatian
Department of Linguistics
Stony Brook University
Stony Brook, NY, USA
hossep.dolatian@stonybrook.edu

Aniello De Santo
Department of Linguistics
University of Utah
Salt Lake City, Utah, USA
aniello.desanto@utah.edu

Thomas Graf
Department of Linguistics
Stony Brook University
Stony Brook, NY, USA
mail@thomasgraf.net

Abstract

This paper investigates bounds on the generative capacity of prosodic processes, by focusing on the complexity of recursive prosody in coordination contexts in English (Wagner, 2010). Although all phonological processes and most prosodic processes are computationally regular string languages, we show that recursive prosody is not. The output string language is instead parallel multiple context-free (Seki et al., 1991). We evaluate the complexity of the pattern over strings, and then move on to a characterization over trees that requires the expressivity of multi bottom-up tree transducers. In doing so, we provide a foundation for future mathematically grounded investigations of the syntax-prosody interface.

1 Introduction

At the level of words, all attested processes in phonology form regular string languages and can be generated via finite-state acceptors (FSAs) and transducers (FSTs) (Johnson, 1972; Kaplan and Kay, 1994; Heinz, 2018). However, not much attention has been given to the generative capacity of prosodic processes at the phrasal or sentential level (but see Yu, 2019). The little work that exists in this respect has shown that many attested intonational processes are finite-state and regular (Pierrehumbert, 1980). It is thus a common hypothesis in the literature that the cross-linguistic typology of prosodic phonology should also be regular.

In this paper, we falsify this hypothesis by providing a mathematically grounded characterization of a pattern of recursive prosody in English coordination, as empirically documented by Wagner (2010). Specifically, we show that when converting a syntactic representation into a prosodic representation, the string language that is generated by this prosodic process is neither a regular nor context-free language, and thus cannot be generated by string-based FSAs. As a tree-to-tree function, the pattern can be captured by a class of bottom-up tree transducers whose outputs correspond to parallel multiple context-free string languages.

This paper is organized as follows. In §2, we provide a literature review of phonology and prosodic phonology, with emphasis on the general tendency for regular computation. In §3, we describe the recursive prosody of coordination structures, and why it cannot be generated with an FST over string inputs. In §4, we show how a multi bottom-up tree transducer can generate the prosodic patterns. We discuss our results in §5, and conclude in §6.

2 Computation of prosody

Within computational prosody, there are two strands of work. One focuses on the generation of prosodic structure at or below the word level. The other operates above the word-level.

At the word level, there is a plethora of work on generating prosodic constituents, all of which require finite-state or regular computation, whether for syllables (Kiraz and Möbius, 1998; Yap, 2006; Hulden, 2006; Idsardi, 2009), feet (van Oostendorp, 1993; Idsardi, 2009; Yu, 2017), or prosodic words (Coleman, 1995; Chew, 2003).[1] In fact, most word-level prosody seems to require at most *subregular* computation (Strother-Garcia, 2018, 2019; Hao, 2020; Dolatian, 2020; Dolatian et al., 2021; Koser, in prep).

However, there is a dearth of formal results for phrasal or intonational prosody. Early work in generative phonology treated the prosodic representations as directly generated from the syntax, with any deviations caused by readjustment rules (Chomsky and Halle, 1968). Notoriously, syntactic representations are at

[1] For syllables and feet, there is a large literature of formalization within Declarative Phonology (Scobbie et al., 1996). This work tends to employ formal representations that are similar to context-free grammars (Klein, 1991; Walther, 1993, 1995; Dirksen, 1993; Coleman, 1991, 1992, 1993, 1996, 2000, 1998; Coleman and Pierrehumbert, 1997; Chew, 2003). But these representations can be restricted enough to be equivalent to regular languages (see earlier such restrictions in Church, 1983).

Proceedings of the Seventeenth SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology,pages 11–22

least context-free (Chomsky, 1956; Chomsky and Schützenberger, 1959). Because sentential prosody interacts with the syntactic level in non-trivial ways, it might seem sensible to assume that 1) the transformation from syntax to prosody is not finite-state definable (= definable with finite-state transducers), and that 2) the string language of prosodic representations is a supra-regular language, not a regular language. Importantly though, this assumption is not trivially true. In fact, early work has shown that even if syntax is context-free, the corresponding prosodic structures can be a regular string language. For instance, Reich (1969) argued that the prosodic structures in SPE can be generated via finite-state devices (see also Langendoen, 1975), while Pierrehumbert (1980) modeled English intonation using a simple finite-state acceptor.

When analyzed over string languages, this mismatch between supra-regular syntax and regular prosody was not explored much in the subsequent literature. In fact, it seems that current research on computational prosody uses the premise that prosodic structures are at most regular (Gibbon, 2001). Crucially, this premise is confounded by the general lack of explicit mathematical formalizations of prosodic systems. For example, there are algorithms for Dutch intonation that capture surface intonational contours and other acoustic cues (t'Hart and Cohen, 1973; t'Hart and Collier, 1975). These algorithms however do not themselves provide sufficient mathematical detail to show that the prosodic phenomenon in question is a regular string language. Instead, one has to deduce that Dutch intonation is regular because the algorithm does not utilize counting or unbounded look-ahead (t'Hart et al., 2006, pg. 114).

As a reflection of this mismatch, early work in prosodic phonology assumed something known as the strict layer hypothesis (SLH; Nespor and Vogel, 1986; Selkirk, 1986). The SLH assumed that prosodic trees cannot be recursive — i.e. a prosodic phrase cannot dominate another prosodic phrase — thus ensuring that a prosodic tree will have fixed depth. Subsequent work in prosodic phonology weakened the SLH: prosodic recursion at the phrase or sentence level is now accepted as empirically robust (Ladd 1986, 2008, ch8; Selkirk 2011; Ito and Mester 2012, 2013). But empirically, it is difficult to find cases of unbounded prosodic recursion (Van der Hulst, 2010). Consider a language that uses only bounded prosodic recursion — e.g. there can be at most two recursive levels of prosodic phrases. The prosodic tree will have fixed depth; and the computation of the corresponding string language is regular. It is then possible to create a computational network that uses a supra-regular grammar for the syntax which interacts with a finite-state grammar for the prosody (Yu and Stabler, 2017; Yu, 2019). To summarize, it seems that the implicit consensus in computational prosody is that 1) syntax can be supra-regular, but the corresponding prosody is regular; 2) prosodic recursion is bounded.

However, as we elaborate in the next section, coordination data from Wagner (2005) is a case where syntactic recursion generates potentially unbounded-recursive prosodic structure. The rest of the paper is then dedicated to exploring the consequences of this construction for the expressivity of sentential prosody.

3 Prosodic recursion in coordination

To our knowledge, Wagner (2005, 2010) is the clearest case where syntactic recursion gets mapped to recursive prosody, such that the recursion is unboundedly deep for the prosody. In this section, we go over the data and generalizations (§3.1), we sketch Wagner's cyclic analysis (§3.2), and we discuss issues with finiteness (§3.3). Finally, we show that that this construction does not correspond to a regular string language (§3.4).

3.1 Unbounded recursive prosody

Wagner documents unbounded prosodic recursion in the coordination of nouns, in contrast to earlier results which reported flat non-recursive prosody (Langendoen, 1987, 1998). Based on experimental and acoustic studies, Wagner reports that recursive coordination creates recursively strong prosodic boundaries. Syntactic edges have a prosodic strength that incrementally depends on their distance from the bottom-most constituents.

When three items are coordinated with two non-identical operators, then two syntactic parses are possible. Each syntactic parse has an analogous prosodic parse. The prosodic parse is based on the relative strength of a prosodic boundary, with | being weaker than ||. The boundary is placed before the operator.

Table 1: Prosody of three items with non-identical operators

Syntactic grouping	Prosodic grouping
[A and [B or C]]	A || and B | or C
[[A and B] or C]	A | and B || or C

When the two operators are identical, then three syntactic and prosodic parses are possible. The

difference between the parses is determined by semantic associativity. For example, a sentence like *I saw [[A and B] and C]* means that I saw A and B together, and I saw C separately.

Table 2: Prosody of three items with identical operators

Syntactic grouping	Prosodic grouping
[A and [B and C]]	A || and B | and C
[[A and B] and C]	A | and B || and C
[[A and B and C]	A | and B | and C

When four items are coordinated, then at most 11 parses are possible. The maximum is reached when the three operators are identical. We can have three levels of prosodic boundaries, ranging from the weakest | to the strongest |||.

Table 3: Prosody of four items with identical operators

Syntactic grouping	Prosodic grouping
[A and B and C and D]	A | and B | and C | and D
[A and B and [C and D]]	A || and B || and C | and D
[A and [B and C] and D]	A || and B | and C || and D
[[A and B] and C and D]	A | and B || and C || and D
[A and [B and C and D]]	A || and B | and C | and D
[[A and B and C] and D]	A | and B | and C || and D
[[A and B] and [C and D]]	A | and B || and C | and D
[A and [B and [C and D]]	A ||| and B || and C | and D
[A and [[B and C] and D]]	A ||| and B | and C || and D
[[A and [B and C]] and D]	A || and B | and C ||| and D
[[[A and B] and C] and D]	A | and B || and C ||| and D

We can extract the following generalizations from the data above. First, the depth of a constituent directly affects the prosodic strength of its edges. At a syntactic edge, the strength of the prosodic boundary depends on the distance between that edge and the most embedded element: for instance, in (1a) the left-bracket between A-B is mapped to a prosodic boundary of strength three |||, because A is above two layers of coordination. The deepest constituent C-D gets the weakest boundary |. Second, when there is associativity, the prosodic strength percolates to other positions within this associative span. For example, in (1b) the boundary of strength || is percolated to A-B from B-C.

1. *Generalizations on coordination*
   (a) *Strength is long-distantly calculated*
   [A and [B and [C and D]]] *is mapped to*
   A ||| and B || and C | and D
   (b) *Strength percolates when associative*
   [A and B and [C and D]] *is mapped to*
   A || and B || and C | and D

3.2 Wagner's cyclic analysis

In order to generate the above forms, Wagner devised a cyclic procedure which we summarize with the algorithm below.

2. *Wagner's cyclic algorithm*
   (a) **Base case**: Let X be a constituent that contains a set of unprosodified nouns (terminal nodes) that are in an associative coordination. Place a boundary of strength | between each noun.
   (b) **Recursive case**: Consider a constituent Y. Let S be a set of constituents S (terminals or non-terminals) that is properly contained in Y, such that at least one constituent in S be prosodified. Let $|^k$ be the strongest prosodic boundary inside Y. Place the boundary $|^{k+1}$ between each constituent in Y.

The algorithm is generalized to coordination of any depth. It takes as input a syntactic tree, and the output is prosodically marked strings. We illustrate this below, with the input tree represented as a bracketed string.

3. *Illustrating Wagner's algorithm*

Input	[A and B and [C and D]]
Base case	C | and D
Recursive case	A || and B || and C | and D

3.3 Issues of finiteness

Because Wagner's study used noun phrases with at most three or four items, the resulting language of prosodic parses is a finite language. Thus, the relevant syntax-to-prosody function is bounded. It is difficult to elicit coordination of 5 items, likely due to processing reasons (Wagner, 2010, 194).

If the primary culprit is performance, though, then syntactic competence may in fact allow for coordination constructions of unbounded depth with any number of items. Wagner's algorithm generates a prosodic structure for any such sentence, such as for (4). For the rest of this paper, we abstract away the finite bounds on coordination size in order to analyze the generative capacity of the underlying system (see Savitch, 1993, for mathematical arguments in support of factoring out finite bounds).

4. *Hypothetical prosody for large coordination*
   [A and B and [C and [D and E]]] *is mapped to*
   A ||| and B ||| and C || and D | and E

3.4 Computing recursive prosody over strings

The choice of representation plays an important role in determining the generative capacity of the prosodic mapping. We first start by treating the mapping as a string-to-string function. We show that the mapping is not regular.

Let the input language be a bracketed string language, such that the input alphabet is a set of nouns{A, ..., Z}, coordinators, and brackets. The output language replaces the brackets with substrings of $|^*$. For illustration, assume that the input language is guaranteed to be a well-bracketed string. At a syntactic boundary, we have to calculate the number of intervening boundaries between it and deepest node. But this requires unbounded memory. For instance, to parse the example below, we incrementally increase the prosodic strength of each boundary as we read the input left-to-right.

5. *Linearly parsing the prosody*:
   [[[A and B] and C] and D] *is mapped to*
   A | and B || and C ||| and D, *where*
   Input alphabet Σ ={ A, ... , Z, and, or, [,]}
   Output alphabet Δ ={ A, ... , Z, and, or, |}
   Input language is Σ^* *and well-bracketed*

Given the above string with only left-branching syntax, the leftmost prosodic boundary will have a juncture of strength |. Every subsequent prosodic boundary will have incrementally larger strength. Over a string, this means we have to memorize the number x of prosodic junctures that were generated at any point in order to then generate $x+1$ junctures at the next point. A 1-way FST cannot memorize an unbounded amount of information. Thus, this function is not rational function and cannot be defined by a 1-way FST. To prove this, we can look at this function in terms of the size of the input and output strings.

6. *Illustrating growth size of recursive prosody*
   [^{n} A_0 and A_1] and A_2] and ... and A_n]
   is mapped to
   A_0 | and A_1 || and A_2 ||| and ... $|^n$ and A_n

Abstractly, for a left-branching input string with n number of left-brackets [, the output string has a monotonically increasing number of prosodic junctures: $| \cdots || \cdots ||| \cdots |n$. The total number of prosodic junctures is a triangular number $n(n+1)/2$. We thus derive the following lemma.

Lemma 1. *For generating coordination prosody as a string-to-string function, the size of the output string grows at a rate of at least* $O(n^2)$ *where* n *is the size of the input string.*

Such a function is neither rational nor regular. Rational functions are computed by 1-way FSTs, and regular functions by 2-way FSTs (Engelfriet and Hoogeboom, 2001).[2] They share the following property in terms of growth rates (Lhote, 2020).

Theorem 1. *Given an input string of size* n*, the size of the output string of a regular function grows at most linearly as* $c \cdot n$*, where* c *is a constant.*

Thus, this string-to-string function is not regular. It could be a more expressive polyregular function (Engelfriet and Maneth, 2002; Engelfriet, 2015; Bojańczyk, 2018; Bojańczyk et al., 2019), a question that we leave for future work.

The discussion in this section focused on generating the output prosodic string when the input syntax is a *bracketed string*. Importantly though, Lemma 1 entails that no matter how one chooses their string encoding of syntactic structure, prosody cannot be modeled as a rational transduction unless there is an upper bound on the minimum number of output symbols that a single syntactic boundary must be rewritten as. To the best of our knowledge, there is no syntactic string encoding that guarantees such a bound. In the next section, we will discuss how to compute prosodic strength starting from a *tree*.

4 Computing recursive prosody over trees

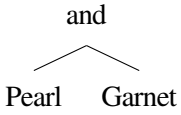
Wagner (2010)'s treatment of recursive prosody assumes an algorithm that maps a syntactic tree to a prosodic string. It is thus valuable to understand the complexity of processes at the syntax-prosody interface starting from the tree representation of a sentence. Assuming we start from trees, there is one more choice to be made, namely whether the prosodic information (in the output) is present within a string or a tree. Notably, every tree-to-string transduction can be regarded as a tree-to-tree transduction plus a string yield mapping. As the tree-to-tree case subsumes the tree-to-string one, it makes sense to consider only the former. For a tree-to-tree mapping, the goal is to obtain a tree representation that already contains the correct prosodic information (Ladd, 1986; Selkirk, 2011). This is the focus of the rest of this paper.

4.1 Dependency trees

When working over syntactic structures explicitly, it is important to commit to a specific tree representation.

[2] This equivalence only holds for functions and deterministic FSTs. Non-deterministic FSTs can also compute relations.

In what follows, we adopt a type of *dependency trees*, where the head of a phrase is treated as the mother of the subtree that contains its arguments. For example, the coordinated noun phrase *Pearl and Garnet* is represented as the following dependency tree.

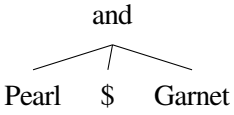


Dependency trees have a rich tradition in descriptive, theoretical, and computational approaches to language, and their properties have been defined across a variety of grammar formalisms (Tesnière, 1965; Nivre, 2005; Boston et al., 2009; Kuhlmann, 2013; Debusmann and Kuhlmann, 2010; De Marneffe and Nivre, 2019; Graf and De Santo, 2019; Shafiei and Graf, 2020, a.o.). Dependency trees keep the relation between heads and arguments local, and they maximally simplify the readability of our mapping rules. Hence, they allow us to focus our discussion on issues that are directly related to the connection of coordinated embeddings and prosodic strength, without having to commit to a particular analysis of coordinate structure.

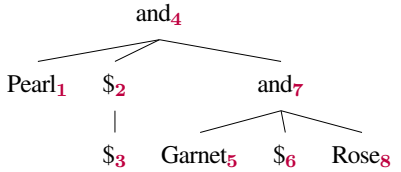
Importantly, this choice does not impact the generalizability of the solution. It is fairly straightforward to convert basic dependency trees into phrase structure trees. Similarly, although it is possible to adopt n-ary branching structures, we chose to limit ourselves to binary trees (in the input). This turns out to be the most conservative assumption, as it forces us to explicitly deal with associativity and flat prosody.

4.2 Encoding prosodic strength over trees

We are interested in the complexity of mapping a "plain" syntactic tree to a tree representation which contains the correct prosodic information. Because of this, we encode prosodic strength over trees in the form of strength boundaries at each level of embedding. Each embedding level in our final tree representation will thus have a prosodic strength branch. The tree below shows how the syntactic tree for *Pearl and Garnet* is enriched with prosodic information, according to our encoding choices. For readability, we use \$ to mark prosodic boundaries in trees instead of |, since the latter could be confused with a unary tree branch.



As the tree below shows, the depth of the prosody branch at each embedding level corresponds to the number of prosodic boundaries needed at that level.



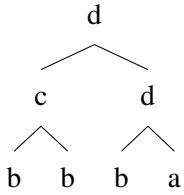
Finally, the prosodic tree is fed to a *yield function* to generate an output prosodified string. In particular, the correct tree-to-string mapping can be obtained by a modified version of a recursive-descent yield, which enumerates nodes left-to-right, depth first, and only enumerates the mother node of each level *after* the boundary branch. This strategy is depicted by the numerical subscripts in the tree above, which reconstruct how the yield of the prosodically annotated tree produces the string: *Pearl* || *and Garnet* | *and Rose*. The rest of this section will focus on how to obtain the correct tree encoding of prosodic information, starting from a plain dependency tree.

4.3 Mathematical preliminaries

For a natural number n, we let $[n] = \{1, \dots, n\}$. A *ranked alphabet* Σ is a finite set of symbols, each one of which has a *rank* assigned by the function $r : \Sigma \to \mathbb{N}$. We write $\Sigma^{(n)}$ to denote $\{\sigma \in \Sigma \mid r(\sigma) = n\}$, and $\sigma^{(n)}$ indicates that σ has rank n.

Given a ranked alphabet Σ and a set A, $T_\Sigma(A)$ is the set of all trees over Σ indexed by A. The symbols in Σ are possible labels for nodes in the tree, indexed by elements in A. The set T_Σ of Σ-*trees* contains all $\sigma \in \Sigma^{(0)}$ and all terms $\sigma^{(n)}(t_1, \dots, t_n)$ $(n \geq 0)$ such that $t_1, \dots, t_n \in T_\Sigma$. Given a term $m^{(n)}(s_1, \dots, s_n)$ where each s_i is a subtree with root d_i, we call m the *mother* of the *daughters* $d_1, \dots, d_n$ $(1 \leq i \leq n)$. If two distinct nodes have the same mother, they are *siblings*. Essentially, the rank of a symbol denotes the finite number of daughters that it can take. Elements of A are considered as additional symbols of rank 0.

Example 1. Given $\Sigma := \{a^{(0)}, b^{(0)}, c^{(2)}, d^{(2)}\}$, T_Σ is an infinite set. The symbol $a^{(0)}$ means that a is a terminal node without daughters, while $c^{(2)}$ is a non-terminal node with two daughters. For example, consider the tree below.



This tree corresponds to the term $d(c(b, b), d(b, a))$, contained in T_Σ. ⌟

As is standard in defining meta-rules, we introduce X as a countably infinite set of variable symbols ($X \cap \Sigma = X$) to be used as place-holders in the definitions of transduction rules over trees.

4.4 Multi bottom-up tree transducers

We assume that the starting point of the prosodic process is a plain syntactic tree. Thus, in order to derive the correct prosodic encoding, we need to propagate information about levels of coordination embedding and about associativity. We adopt a bottom-up approach, and characterize this process in terms of *multi bottom-up tree transducers* (MBOT; Engelfriet et al., 1980; Lilin, 1981; Maletti, 2011). Essentially, MBOTs generalize traditional bottom–up tree transducers in that they allow states to pass more than one output subtree up to subsequent transducer operations (Gildea, 2012). In other words, each MBOT rule potentially specifies several parts of the output tree. This is highlighted by the fact that the transducer states ($q \in Q$) can have rank greater than one — i.e. they can have more than one daughter, where the additional daughters are used to hold subtrees in *memory*. We follow Fülöp et al. (2004) in presenting the semantics of MBOTs.

Definition 1 (MBOT). A multi bottom-up tree transducer (MBOT) is a tuple $M = (Q, \Sigma, \Delta, \mathit{root}, q_f, R)$, where Q, $\Sigma \cup \Delta$, $\{\mathit{root}\}$, $\{q_f\}$ are pairwise disjoint, such that:

- Q is a ranked alphabet with $Q^{(0)} = \emptyset$, called the set of states
- Σ and Δ are ranked input and output alphabets, respectively
- *root* is a unary symbol, called the root symbol
- q_f is a unary symbol called the final state

R is a finite set of rules of two forms:

- $\sigma(q_1(x_{1,1},\ldots,x_{1,n_1}),\ldots,q_k(x_{k,1},\ldots,x_{k,n_k})) \rightarrow q_0(t_1,\ldots,t_{n_0})$

where $k \geq 0$, $\sigma \in \Sigma^{(k)}$, for every $i \in [k] \cup \{0\}$, $q_i \in Q^{(n_i)}$ for some $n_i \geq 1$, for every $j \in [n_0], t_j \in T_\Delta(\{x_{i,j} | i \in [k], j \in [n_i]\})$.

- $\mathit{root}(q(x_1,\ldots,x_n)) \rightarrow q_f(t)$

where $n \geq 1, q \in Q^{(n)}$, and $t \in T_\Delta(X_n)$. ⌟

The derivational relation induced by M is a binary relation $\Rightarrow_M$ over the set $T_{\Sigma \cup \Delta \cup Q \cup \{root, q_f\}}$ defined as follows. For every $\varphi, \psi \in T_{\Sigma \cup \Delta \cup Q \cup \{root, q_f\}}$, $\varphi \Rightarrow_M \psi$ iff there is a tree $\beta \in T_{\Sigma \cup \Delta \cup Q \cup \{root, q_f\}}(X_1)$ s.t. x_1 occurs exactly once in β and either there is a rule

- $\sigma(q_1(x_{1,1},\ldots,x_{1,n_1}),\ldots,q_k(x_{k,1},\ldots,x_{k,n_k})) \rightarrow r$ in R

and there are trees $T_{i,j} \in T_\Sigma$ for every $i \in [k]$ and $j \in [n_i]$, s.t. $\varphi = \beta[\sigma(q_1(t_{1,1}, \ldots, t_{1,n_1}), \ldots, q_k(t_{k,1}, \ldots, t_{k,n_k}))]$, and $\psi = \beta[r[x_{i,j} \leftarrow t_{i,j} | i \in [k], j \in [n_i]]]$; or there is a rule

- $\mathit{root}(q(x_1,\ldots,x_n)) \rightarrow q_f(t)$ in R

and there are trees $t_i \in T_\Delta$ for every $i \in [n]$ s.t. $\varphi = \beta[\mathit{root}(q(t_1,\ldots,t_n))]$, and $\psi = \beta[q_f(t[t_1,\ldots,t_n])]$. The tree transformation computed by M is the relation:

$$\tau_M = \{(s,t) \in T_\Sigma \times T_\Delta \mid \mathit{root}(s) \Rightarrow_M^* q_f(t)\}$$

Intuitively, tree transductions are performed by rewriting a local tree fragment as specified by one of the rules in R. For instance, a rule can replace a subtree, or copy it to a different position. Rules apply bottom–up from the leaves of the input tree, and terminate in an accepting state q_f.

4.5 MBOT for recursive prosody

We want a transducer which captures Wagner (2010)'s bottom-up cyclic procedure. Consider now the MBOT $M_{pros} = (Q, \Sigma, \Delta, \mathit{root}, q_f, R)$, with $Q = \{q_*, q_c\}$, $\sigma_c \in \{\mathit{and}, \mathit{or}\} \subsetneq \Sigma$, $\sigma \in \Sigma - \{\mathit{and}, \mathit{or}\}$, and $\Sigma = \Delta$. We use q_c to indicate that M_{pros} has verified that a branch contains a coordination (so σ_c), with q_* assigned to any other branch. As mentioned, we use \$ to mark prosodic boundaries in the trees instead of |. The set of rules R is as follows.

Rule 1 rewrites a terminal symbol σ as itself. The MBOT for that branch transitions to $q_*(\sigma)$.

$$\sigma \rightarrow q_*(\sigma) \tag{1}$$

Rule 2 applies to a subtree headed by $\sigma_c \in \{\mathit{and}, \mathit{or}\}$, with only terminal symbols as daughters: $\sigma_c(q_*(x), q_*(y))$. It inserts a prosodic boundary \$ between the daughters x, y. The boundary \$ is also copied as a daughter of the mother q_c, as record of the fact that we have seen one coordination level.

$$\sigma_c(q_*(x), q_*(y)) \rightarrow q_c(\sigma_c(x, \$, y), \$) \tag{2}$$

We illustrate this in Figure 1 with a coordination of two items, representing the mapping: [B and A] $\rightarrow$ *B | and A*. We also assume that sentence-initial boundaries are vacuously interpreted.

We now consider cases where a coordination is the mother not just of terminal nodes, but of other coordinated phrases. Rule 3 handles the case in which

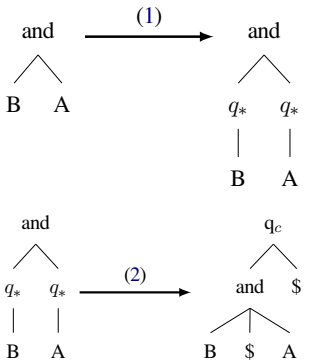


Figure 1: Example of the application of rules (1) and (2). The numerical label on the arrow indicates which rule was applied in order to rewrite the tree on the left as the tree on the right.

the right sibling of the mother was also headed by a coordination (as encoded by σ_c having q_c as one of its daughters). Here, q_c is the result of a previous rule application (e.g. rule 2) and it has two subtrees itself: $q_c(w,y)$. Although we do not have access to the internal labels of x, y, and w, by the format of the previous rules we know that the right daughter of q_c (i.e. y) is the one that contains the strength information. Then, rule 3 has three things to do. It increments y by one boundary: $\$(y)$. It places $\$(y)$ in between the two subtrees x and w. And, it copies $\$(y)$ as the daughter of the new q_c state in order to propagate $\$(y)$ to the next embedding level (see Figure 2).

$$\sigma_c(q_*(x),q_c(w,y)) \rightarrow q_c(\sigma_c(x,\$(y),w),\$(y)) \quad (3)$$

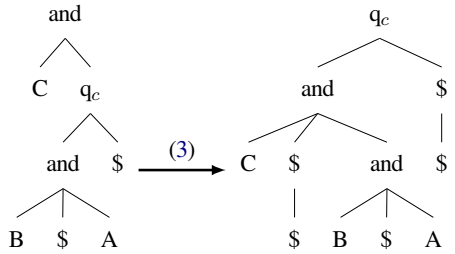


Figure 2: Example of the application of rule (3). For ease of readability, we omit q_* states over terminal nodes.

Rule 4 applies once all coordinate phrases up to the root have been rewritten. It simply rewrites the root as the final accepting state. It gets rid of the daughter of q_c that contains the strength markers, since there is no need to propagate them any further.

$$root(q_c(x,y)) \rightarrow q_f(x) \quad (4)$$

As the examples so far should have clarified, M_{pros} as currently defined readily handles cases where the embedding of the coordination is strictly right branching, with the bulk of the work done via rule 3. However, while these rules work well for instances in which a coordination is always the right daughter of a node, they cannot deal with cases in which the coordination branches left, or alternates between the two. This is easily fixed by introducing variants to rule 3, which consider the position of the coordination as marked by q_c. Importantly, the position of the copy of the boundary branch is not altered, and it is always kept as the rightmost sibling of q_c. What changes is the relative position of the w and x subbranches in the output (see Figure 3).

$$\sigma_c(q_c(w,y),q_*(x)) \rightarrow q_c(\sigma_c(w,\$(y),x),\$(y)) \quad (5)$$

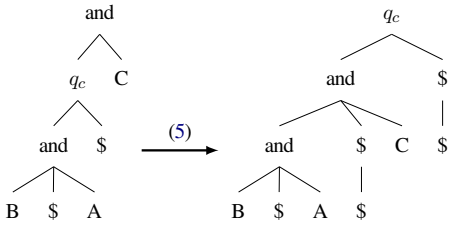


Figure 3: Left branching example as in rule (5).

Following the same logic, rule 6 handles cases like *[[A and B] and [C and D]]*, in which both daughters of a coordination are headed by a coordination themselves (see Figure 4).

$$\sigma_c(q_c(x,z),q_c(y,w)) \rightarrow q_c(\$(x),\sigma_c(z,\$(x),w)) \quad (6)$$

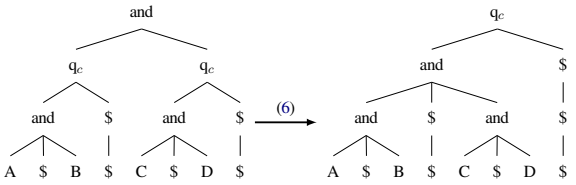


Figure 4: Example of the application of rule (6).

Finally, we need to take care of the flat prosody or associativity issue. The MBOT M_{pros} as outlined so far increases the depth of the boundary branch at each level of embedding. Because we are adopting binary branching trees, the current set of rules is trivially unable to encode cases like *[A and B and C]*. We follow Wagner's assumption that semantic information on the syntactic tree guides the prosody cycles. Representationally, we mark this by using specific labels on the internal nodes of the tree. We assume that the flat constituent interpretation is

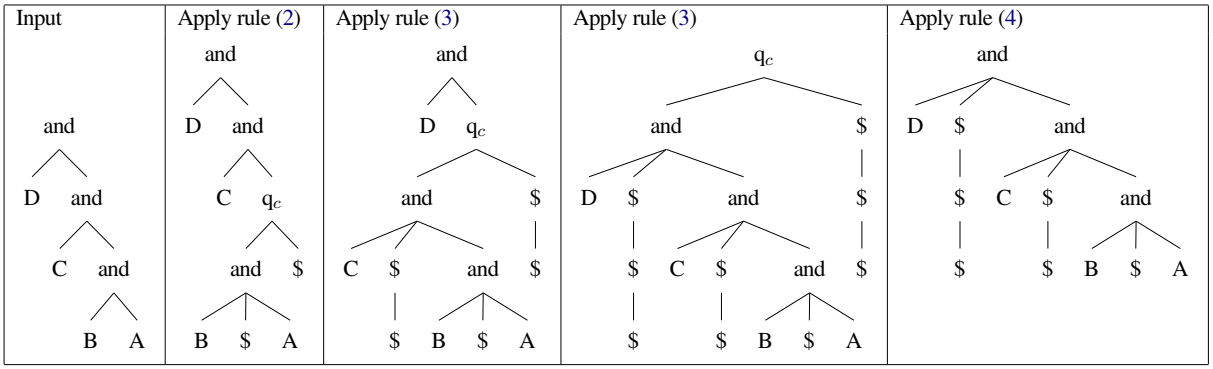


Figure 5: Walk-through of the transduction defined by M_{pros}. For ease of readability, and to highlight how q_c propagates embedding information about the coordination, q_* and q_f states are omitted.

obtained by marking internal nodes as non-cyclic, introducing the alphabet symbol σ_n:

$$\sigma_n(q_*(x),q_c(w,y)) \rightarrow q_c(\sigma_c(x,y,w),y) \qquad (7)$$

Essentially, rule 7 tells us that when a coordination node is marked as σ_n, M_{pros} just propagates the level of prosodic strength that it currently has registered (in y), without increments (see Figure 6). This rule can be trivially adjusted to deal with branching differences, as done for rules 3 and 5.

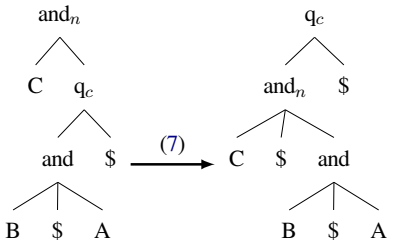


Figure 6: Application of rule (7) for flat prosody.

A full, step by step M_{pros} transduction is shown in Figure 5. Taken together, the recursive prosodic patterns are fully characterized by M_{pros} when it is adjusted with a set of rules to deal with alternating branching and flat associativity. The tree transducer generates tree representations where each level of embedding is marked by a branch, which carries information about the prosodic strength for that level. As outlined in Section 4.2, this final representation may then be fed to a modified string yield function for dependency tree languages.

Dependency trees allowed us to present a transducer with rules that are relatively easy to read. But, as mentioned before, this choice does not affect our general result. Under the standard assumption that the distance between the head of a phrase and its maximal projection is bounded, M_{pros} can be extended to phrase structure trees, by virtue of the bottom-up strategy being intrinsically equipped with finite look-ahead. A switch to phrase structure trees may prove useful for future work on the interaction of prosody and movement.

5 Generating recursive prosody

The previous section characterized recursive prosody over trees with a non-linear, deterministic MBOT. This is a nice result, as MBOTs are generally well-understood in terms of their algorithmic properties. Moreover, this result is in line with past work exploring the connections of MBOTs, tree languages, and the complexity of movement and copying operations in syntax (Kobele, 2006; Kobele et al., 2007, a.o.).

We can now ask what the complexity of this approach is. MBOTs generate output string languages that are potentially parallel multiple context-free languages (PMCFL; Seki et al., 1991, 1993; Gildea, 2012; Maletti, 2014; Fülöp et al., 2005). Since this class of string languages is more powerful than context-free, the corresponding tree language is not a regular tree language (Gécseg and Steinby, 1997). This is not surprising, as MBOTs can be understood as an extension of synchronous tree substitution grammars (Maletti, 2014).

Notably, independently of our specific MBOT solution, prosody as defined in this paper generates at least some output string languages that lack the constant growth property — hence, that are PMCFLs. Consider as input a regular tree language of left-branching coordinationate phrases, where each level is simply of the form *and(X, Mary)*. The $n-th$ level of embedding from the top extends the string yield by $n+2$ symbols. This immediately implies no constant growth, and thus no semi-linearity (Weir, 1988; Joshi et al., 1990).

Interestingly though, the prosody MBOT developed here is fairly limited in its expressivity as the

transducer states themselves do almost no work, and most of the transduction rules in M_{pros} rely on the ability to store the prosody strength branch. Hence, the specific MBOT in this paper might turn out to belong to a relatively weak subclass of tree transductions with copying, perhaps a variant of input strictly local tree transductions (cf. Ikawa et al., 2020; Ji and Heinz, 2020), or a transducer variant of sensing tree automata (cf. Fülöp et al., 2004; Kobele et al., 2007; Maletti, 2011, 2014; Graf and De Santo, 2019). Since all of those have recently been used in the formal study of syntax, they are natural candidates for a computational model of prosody, and their sensitivity to minor representational difference might also illuminate what aspects of syntactic representation affect the complexity of prosodic processes.

Finally, one might worry that the mathematical complexity is a confound of the representation we use, rather than a genuine property of the phenomenon. However, a representation of prosodic strength is necessary and cannot be reduced further for two reasons. First, strength cannot be reduced to syntactic boundaries because a single prosodic edge (may correspond to $|^k$ for any $k \geq 1$. As discussed in depth by Wagner (2005, 2010), one cannot simply convert a syntactic tree into a prosodic tree by replacing the labels of nonterminal nodes. Second, strength also cannot be reduced to different categories of prosodic constituents — e.g. assuming that | is a prosodic phrase while || is an intonational phrase. As argued in depth in (Wagner, 2005, 2010), these different constituent types do not map neatly to prosodic strength. Instead, these boundaries all encode relative strengths of prosodic phrase boundaries.

6 Conclusion

This paper formalizes the computation of unbounded recursive prosodic structures in coordination. Their computation cannot be done by string-based finite-state transducers. They instead need more expressive grammars. To our knowledge, this paper is one of the few (if only) formal results on how prosodic phonology at the sentence-level is computationally more expressive than phonology at the word-level.

As discussed above, recent work in prosodic phonology relies on the assumption that prosodic structure can be recursive. However, because such work usually uses bounded-recursion, such phenomena are computationally regular. Departing from this stance, this paper focused on the prosodic phenomena reported in Wagner (2005) as a core case study, because of the following fundamental properties:

- The syntax has unbounded recursion.
- The prosody has unbounded recursion.
- All recursive prosodic constituents have the same prosodic label (= a prosodic phrase).
- The recursive prosodic constituents have acoustic cues marking different strengths.
- There is an algorithm which explicitly assigns the recursive prosodic constituents to these different strengths.

In this paper, we focused on explicitly generating the prosodic strengths at each recursive prosodic levels, putting aside the mathematically simpler task of converting a recursive syntactic tree into a recursive prosodic tree (Elfner, 2015; Bennett and Elfner, 2019) — which is a process essentially analogous to a relabeling of the nonterminal nodes of the syntactic tree, without care for the prosodic strength. The mapping studied in this paper has been conjectured in the past to be computationally more expressive than regular languages or functions (Yu and Stabler, 2017). Here, we formally verified that hypothesis.

An open question then is to find other empirical phenomena which also have the above properties. One potential area of investigation is the assignment of relative prominence relations in English compound prosody (Chomsky and Halle, 1968). However, English compound prosody is a highly controversial area. It is unclear what is the current consensus on an *exact* algorithm for these compounds, especially one that utilizes recursion and is not based on impressionistic judgments (Liberman and Prince, 1977; Gussenhoven, 2011). In this sense, the mathematical results in this paper highlight the importance of representational commitments and of explicit assumptions in the study of prosodic expressivity. Our paper might then help identify crucial issues in future theoretical and empirical investigations of the syntax-prosody interface.

Acknowledgements

We are grateful to our anonymous reviewers, Jon Rawski, and Kristine Yu. Thomas Graf is supported by the National Science Foundation under Grant No. BCS-1845344.

References

Ryan Bennett and Emily Elfner. 2019. The syntax–prosody interface. *Annual Review of Linguistics*, 5:151–171.

Mikołaj Bojańczyk. 2018. Polyregular functions. *arXiv preprint arXiv:1810.08760*.

Mikołaj Bojańczyk, Sandra Kiefer, and Nathan Lhote. 2019. String-to-string interpretations with polynomial-size output. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, Patras, Greece. (LIPIcs)*, volume 132, page 106:1–106:14, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

Marisa Ferrara Boston, John T. Hale, and Marco Kuhlmann. 2009. Dependency structures derived from minimalist grammars. In *The Mathematics of Language*, pages 1–12. Springer.

Peter Chew. 2003. *A computational phonology of Russian*. Universal-Publishers, Parkland, FL.

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

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

Noam Chomsky and Marcel P Schützenberger. 1959. The algebraic theory of context-free languages. In *Studies in Logic and the Foundations of Mathematics*, volume 26, pages 118–161. Elsevier.

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

John Coleman. 1992. The phonetic interpretation of headed phonological structures containing overlapping constituents. *Phonology*, 9(1):1–44.

John Coleman. 1993. English word-stress in unification-based grammar. In T. Mark Ellison and James Scobbie, editors, *Computational Phonology*, page 97–106. Centre for Cognitive Science, University of Edinburgh.

John Coleman. 1995. Declarative lexical phonology. In Jacques Durand and Francsis Katamba, editors, *Frontiers of phonology: Atoms, structures, derivations*, pages 333–383. Longman, London.

John Coleman. 1996. Declarative syllabification in Tashlhit Berber. In Jacques Durand and Bernard Laks, editors, *Current trends in phonology: Models and methods*, volume 1, pages 175–216. European Studies Research Institute, University of Salford, Salford.

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

John Coleman. 2000. Candidate selection. *The Linguistic Review*, 17(2-4):167–180.

John Coleman and Janet Pierrehumbert. 1997. Stochastic phonological grammars and acceptability. In *Third meeting of the ACL special interest group in computational phonology: Proceedings of the workshop*, pages 49–56, East Stroudsburg, PA. Association for computational linguistics.

John S Coleman. 1991. Prosodic structure, parameter-setting and ID/LP grammar. In Steven Bird, editor, *Declarative perspectives on phonology*, pages 65–78. Centre for Cognitive Science, University of Edinburgh.

Marie-Catherine De Marneffe and Joakim Nivre. 2019. Dependency grammar. *Annual Review of Linguistics*, 5:197–218.

Ralph Debusmann and Marco Kuhlmann. 2010. Dependency grammar: Classification and exploration. In *Resource-adaptive cognitive processes*, pages 365–388. Springer.

Arthur Dirksen. 1993. Phrase structure phonology. In T. Mark Ellison and James Scobbie, editors, *Computational Phonology*, page 81–96. Centre for Cognitive Science, University of Edinburgh.

Hossep Dolatian. 2020. *Computational locality of cyclic phonology in Armenian*. Ph.D. thesis, Stony Brook University.

Hossep Dolatian, Nate Koser, Kristina Strother-Garcia, and Jonathan Rawski. 2021. Computational restrictions on iterative prosodic processes. In *Proceedings of the 2019 Annual Meeting on Phonology*. Linguistic Society of America.

Emily Elfner. 2015. Recursion in prosodic phrasing: Evidence from Connemara Irish. *Natural Language & Linguistic Theory*, 33(4):1169–1208.

Joost Engelfriet. 2015. Two-way pebble transducers for partial functions and their composition. *Acta Informatica*, 52(7-8):559–571.

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

Joost Engelfriet and Sebastian Maneth. 2002. Two-way finite state transducers with nested pebbles. In *International Symposium on Mathematical Foundations of Computer Science*, pages 234–244. Springer.

Joost Engelfriet, Grzegorz Rozenberg, and Giora Slutzki. 1980. Tree transducers, l systems, and two-way machines. *Journal of Computer and System Sciences*, 20(2):150–202.

Zoltán Fülöp, Armin Kühnemann, and Heiko Vogler. 2004. A bottom-up characterization of deterministic top-down tree transducers with regular look-ahead. *Information Processing Letters*, 91(2):57–67.

Zoltán Fülöp, Armin Kühnemann, and Heiko Vogler. 2005. Linear deterministic multi bottom-up tree transducers. *Theoretical computer science*, 347(1-2):276–287.

Ferenc Gécseg and Magnus Steinby. 1997. Tree languages. In *Handbook of formal languages*, pages 1–68. Springer.

Dafydd Gibbon. 2001. Finite state prosodic analysis of African corpus resources. In *EUROSPEECH 2001 Scandinavia, 7th European Conference on Speech Communication and Technology, 2nd INTERSPEECH Event, Aalborg, Denmark, September 3-7, 2001*, pages 83–86. ISCA.

Daniel Gildea. 2012. On the string translations produced by multi bottom–up tree transducers. *Computational Linguistics*, 38(3):673–693.

Thomas Graf and Aniello De Santo. 2019. Sensing tree automata as a model of syntactic dependencies. In *Proceedings of the 16th Meeting on the Mathematics of Language*, pages 12–26, Toronto, Canada. Association for Computational Linguistics.

Carlos Gussenhoven. 2011. Sentential prominence in English. In Marc van Oostendorp, Colin Ewen, Elizabeth Hume, and Keren Rice, editors, *The Blackwell companion to phonology*, volume 5, pages 1–29. Wiley-Blackwell, Malden, MA.

Yiding Hao. 2020. Metrical grids and generalized tier projection. In *Proceedings of the Society for Computation in Linguistics*, volume 3.

Jeffrey Heinz. 2018. The computational nature of phonological generalizations. In Larry Hyman and Frans Plank, editors, *Phonological Typology*, Phonetics and Phonology, chapter 5, pages 126–195. Mouton de Gruyter, Berlin.

Mans Hulden. 2006. Finite-state syllabification. In Anssi Yli-Jyrä, Lauri Karttunen, and Juhani Karhumäki, editors, *Finite-State Methods and Natural Language Processing. FSMNLP 2005. Lecture Notes in Computer Science*, volume 4002. Springer, Berlin/Heidelberg.

Harry Van der Hulst. 2010. A note on recursion in phonology recursion. In Harry Van der Hulst, editor, *Recursion and human language*, pages 301–342. Mouton de Gruyter, Berlin & New York.

William J Idsardi. 2009. Calculating metrical structure. In Eric Raimy and Charles E. Cairns, editors, *Contemporary views on architecture and representations in phonology*, number 48 in Current Studies in Linguistics, pages 191–211. MIT Press, Cambridge, MA.

Shiori Ikawa, Akane Ohtaka, and Adam Jardine. 2020. Quantifier-free tree transductions. *Proceedings of the Society for Computation in Linguistics*, 3(1):455–458.

Junko Ito and Armin Mester. 2012. Recursive prosodic phrasing in Japanese. In Toni Borowsky, Shigeto Kawahara, Shinya Takahito, and Mariko Sugahara, editors, *Prosody matters: Essays in honor of Elisabeth Selkirk*, pages 280–303. Equinox Publishing, London.

Junko Ito and Armin Mester. 2013. Prosodic subcategories in Japanese. *Lingua*, 124:20–40.

Jing Ji and Jeffrey Heinz. 2020. Input strictly local tree transducers. In *International Conference on Language and Automata Theory and Applications*, pages 369–381. Springer.

C Douglas Johnson. 1972. *Formal aspects of phonological description*. Mouton, The Hague.

Aravind K Joshi, K Vijay Shanker, and David Weir. 1990. The convergence of mildly context-sensitive grammar formalisms. *Technical Reports (CIS)*, page 539.

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

George Anton Kiraz and Bernd Möbius. 1998. Multilingual syllabification using weighted finite-state transducers. In *The third ESCA/COCOSDA workshop (ETRW) on speech synthesis*.

Ewan Klein. 1991. Phonological data types. In Steven Bird, editor, *Declarative perspectives on phonology*, pages 127–138. Centre for Cognitive Science, University of Edinburgh.

Gregory M. Kobele, Christian Retoré, and Sylvain Salvati. 2007. An automata-theoretic approach to minimalism. *Model theoretic syntax at 10*, pages 71–80.

Gregory Michael Kobele. 2006. *Generating Copies: An investigation into structural identity in language and grammar*. Ph.D. thesis, University of California, Los Angeles.

Nate Koser. in prep. *The computational nature of stress assignment*. Ph.D. thesis, Rutgers University.

Marco Kuhlmann. 2013. Mildly non-projective dependency grammar. *Computational Linguistics*, 39(2):355–387.

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

D. Robert Ladd. 2008. *Intonational phonology*. Cambridge University Press, Cambridge.

D. Terence Langendoen. 1975. Finite-state parsing of phrase-structure languages and the status of readjustment rules in grammar. *Linguistic Inquiry*, 6(4):533–554.

D. Terence Langendoen. 1987. On the phrasing of coordinate compound structures. In Brian Joseph and Arnold Zwicky, editors, *A festschrift for Ilse Lehiste*, page 186–196. Ohio State University, Ohio.

D. Terence Langendoen. 1998. Limitations on embedding in coordinate structures. *Journal of Psycholinguistic Research*, 27(2):235–259.

Nathan Lhote. 2020. Pebble minimization of polyregular functions. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 703–712.

Mark Liberman and Alan Prince. 1977. On stress and linguistic rhythm. *Linguistic inquiry*, 8(2):249–336.

Eric Lilin. 1981. Propriétés de clôture d'une extension de transducteurs d'arbres déterministes. In *Colloquium on Trees in Algebra and Programming*, pages 280–289. Springer.

Andreas Maletti. 2011. How to train your multi bottom-up tree transducer. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 825–834.

Andreas Maletti. 2014. The power of regularity-preserving multi bottom-up tree transducers. In *International Conference on Implementation and Application of Automata*, pages 278–289. Springer.

Marina Nespor and Irene Vogel. 1986. *Prosodic phonology*. Foris, Dordrecht.

Joakim Nivre. 2005. Dependency grammar and dependency parsing. *MSI report*, 5133.1959:1–32.

Marc van Oostendorp. 1993. Formal properties of metrical structure. In *Sixth Conference of the European Chapter of the Association for Computational Linguistics*, pages 322–331, Utrecht. ACL.

Janet Breckenridge Pierrehumbert. 1980. *The phonology and phonetics of English intonation*. Ph.D. thesis, Massachusetts Institute of Technology.

Peter. A. Reich. 1969. The finiteness of natural language. *Language*, 45:831–843.

Walter J Savitch. 1993. Why it might pay to assume that languages are infinite. *Annals of Mathematics and Artificial Intelligence*, 8(1-2):17–25.

James M. Scobbie, John S. Coleman, and Steven Bird. 1996. Key aspects of declarative phonology. In Jacques Durand and Bernard Laks, editors, *Current Trends in Phonology: Models and Methods*, volume 2. European Studies Research Institute, Salford, Manchester.

Hiroyuki Seki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. 1991. On multiple context-free grammars. *Theoretical Computer Science*, 88(2):191–229.

Hiroyuki Seki, Ryuichi Nakanishi, Yuichi Kaji, Sachiko Ando, and Tadao Kasami. 1993. Parallel multiple context-free grammars, finite-state translation systems, and polynomial-time recognizable subclasses of lexical-functional grammars. In *Proceedings of the 31st annual meeting on Association for Computational Linguistics*, pages 130–139. Association for Computational Linguistics.

Elisabeth Selkirk. 1986. On derived domains in sentence phonology. *Phonology Yearbook*, 3(1):371–405.

Elisabeth Selkirk. 2011. The syntax-phonology interface. In John Goldsmith, Jason Riggle, and Alan C. L. Yu, editors, *The Handbook of Phonological Theory*, 2 edition, pages 435–483. Blackwell, Oxford.

Nazila Shafiei and Thomas Graf. 2020. The subregular complexity of syntactic islands. In *Proceedings of the Society for Computation in Linguistics*, volume 3.

Kristina Strother-Garcia. 2018. Imdlawn Tashlhiyt Berber syllabification is quantifier-free. In *Proceedings of the Society for Computation in Linguistics*, volume 1, pages 145–153.

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

Lucien Tesnière. 1965. Eléments de syntaxe structurale, 1959. *Paris, Klincksieck*.

Johan t'Hart and Antonie Cohen. 1973. Intonation by rule: a perceptual quest. *Journal of Phonetics*, 1(4):309–327.

Johan t'Hart and René Collier. 1975. Integrating different levels of intonation analysis. *Journal of Phonetics*, 3(4):235–255.

Johan t'Hart, René Collier, and Antonie Cohen. 2006. *A perceptual study of intonation: An experimental-phonetic approach to speech melody*. Cambridge University Press.

Michael Wagner. 2005. *Prosody and recursion*. Ph.D. thesis, Massachusetts Institute of Technology.

Michael Wagner. 2010. Prosody and recursion in coordinate structures and beyond. *Natural Language & Linguistic Theory*, 28(1):183–237.

Markus Walther. 1993. Declarative syllabification with applications to German. In T. Mark Ellison and James Scobbie, editors, *Computational Phonology*, pages 55–79. Centre for Cognitive Science, University of Edinburgh.

Markus Walther. 1995. A strictly lexicalized approach to phonology. In *Proceedings of DGfS/CL'95*, page 108–113, Düsseldorf. Deutsche Gesellschaft für Sprachwissenschaft, Sektion Computerlinguistik.

David Jeremy Weir. 1988. *Characterizing mildly context-sensitive grammar formalisms*. Ph.D. thesis, University of Pennsylvania.

Ngee Thai Yap. 2006. *Modeling syllable theory with finite-state transducers*. Ph.D. thesis, University of Delaware.

Kristine M. Yu. 2017. Advantages of constituency: Computational perspectives on Samoan word prosody. In *International Conference on Formal Grammar 2017*, page 105–124, Berlin. Spring.

Kristine M. Yu. 2019. Parsing with minimalist grammars and prosodic trees. In Robert C. Berwick and Edward P. Stabler, editors, *Minimalist Parsing*, pages 69–109. Oxford University Press, London.

Kristine M. Yu and Edward P. Stabler. 2017. (In) variability in the Samoan syntax/prosody interface and consequences for syntactic parsing. *Laboratory Phonology: Journal of the Association for Laboratory Phonology*, 8(1):1–44.