The Logic of Linearization: Interpretations of Trees via Strings

Vincent Czarnecki
Rutgers University
vincent.czarnecki@rutgers.edu

Abstract

This paper introduces a novel method of linearization, casting it as a model-theoretic *interpretation*. Within Model Theory, an interpretation is a way of understanding a structure through the lens of another structure– in this sense, linearization is an interpretation of a tree's string yield through the lens of the tree. Such a formal characterization allows us to explicitly codify locality into the post-syntax (in line with Embick and Noyer (1999)). This has strong potential implications for the nature of syntax-phonology interaction in terms of formal complexity and typological predictions of phrasal phonology. Crucially, casting linearization in this way also opens the door for a closer unification of how we understand the computational properties of interfaces between linguistic modules more generally.

1 Introduction

Model Theory is a subfield within mathematical logic that is used to formally reason about structures and the properties they satisfy. There has been a rich tradition of using Model Theory within generative semantics. More recently however, research in theoretical computational linguistics has shown that Model Theory is an extremely useful tool for understanding syntax, phonology, morphology, and phonetics as well. Due to Model Theory's abstract and domain-general nature, there is a great deal of freedom in the sorts of structures that can be defined and the mappings between them, making it well-suited for linguistic theorizing.

For example, Model Theory has been used in syntax to formally reason about the computational properties of Government and Binding Theory (Rogers and Nordlinger, 1998). More recently, Model Theory has been used extensively by phonologists to understand both phonological well-formedness of structures (Strother-Garcia et al., 2016; Jardine, 2017) as well as mappings between underlying structures and surface structures (Oakden, 2021; Bhaskar et al., 2020). In Nelson (2024), model-theoretic interpretations are used to model autosegmental coupling graphs, as well as transformations between them and string representations, showing a use case in the phonetics-phonology interface. In (Petrovic, 2023), Model Theory is used to reason about the computational nature of morphological processes. In terms of complexity, this type of formalization also allows for a richer understanding of the tight relationship between learnability and computational simplicity with respect to typological predictions (Lambert et al., 2021; Rawski, 2021).

Knowing that model-theoretic representations have given novel insights to our formal understanding of separate linguistic modules, a natural question arises: *How can we use knowledge of these modules independently to understand their interaction?* Namely, if model-theoretic representations allow us to understand the formal properties of semantics, syntax, phonology, morphology, phonetics in isolation, and we know that it is extremely well-suited for understanding the relationships between different structures, then it should also serve as an invaluable tool for understanding the formal properties of their interfaces. This paper is a step in this direction, showing that linearization can be understood as an interpretation of linear post-syntactic representations through the lens of hierarchical syntactic representations. While this is one particular use case for the much broader endeavor of using Model Theory investigations of the interfaces, this opens up the door for a great body of research while making novel observations about the nature of linearization.

The paper is organized as follows. Section 2 gives an introduction to Model Theory, discussing string models and interpretations. In Section 3, we

Proceedings of the Society for Computation in Linguistics (SCiL) 2025, pages 123-132.
Eugene, Oregon, July 18-20, 2025

discuss linearization and show how it can be formulated as an interpretation from trees to strings and sketches an approach toward incorporating a simple case of movement into the analysis. Section 4 discusses some broader theoretical implications for this view of linearization.

2 Model Theory

A *signature* $\mathcal{S}$ is simply a collection of functions, relations and constants. The discussion here will be limited to dealing with relations, so we will stick to signatures that contain only relations, not functions or constants. A *relational model* is a pair $\langle D \mid r_1, \ldots r_n \rangle$ where D is some domain, and each r_i is a k-ary relation from the signature $\mathcal{S}$ over elements in the domain D. In place of *model*, the word *structure* is also commonly used. Here, k-ary simply means that the relation r_i takes k elements of D as its arguments. For example, $p(x)$ where $x \in D$ would be a unary relation, $q(x, y)$ where $x, y \in D$ would be a binary relation, etc. The focus of this section is on using these models to define strings and mappings between them.

2.1 Strings

Consider the string *apba*. It contains only the segments {*a,b,p*} and there are four elements, the first bearing *a*, the second bearing *p*, the third bearing *b*, and the fourth bearing *a*. Let the domain $D = \{0, 1, 2, 3\}$ represent the indices of the string and the alphabet (set of symbols) $\Sigma = \{a, b, p\}$ represent the labels each index can bear. For each of these symbols, define a unary relation that indicates whether or not an index x of the string bears that symbol: so there are three unary relations $a(x), b(x), p(x)$. For our string *apba*, it is the case that $a(0)$, $p(1)$, $b(2)$, and $a(3)$ are all true, and any of these relations for other domain elements will be false. Each index bears a label, but there must be some way to tell what precedes what in our string. This can be done by relating the indices through a binary precedence relation. Strict precedence $\lhd(x, y)$ states that x comes before y with nothing else in between. A string model for *apba* using strict precedence $\lhd(x, y)$ is shown below:

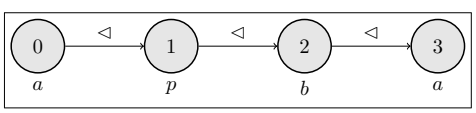

Figure 1: String Model with Strict Precedence

Alternatively, one could use general precedence $< (x, y)$ where x comes before y at any point in the string. Whether a structure is defined using general precedence or strict precedence directly affects the sorts of generalizations one can make. For example, using strict precedence it is natural to ban immediately adjacent segments like a ban on any obstruent immediately following a nasal (say, a *NT constraint), whereas using general precedence it is natural to ban sequences of segments like long distance sibilant harmony that bans an ʃ following an s anywhere in the string (say, a *s ... ʃ constraint). For a broader picture of how representations and the nature of constraints relate to one another in phonology, see Heinz (2018). This difference will have important implications for the motivation of the view of linearization argued for here.

2.2 Interpretations

Informally, a logical interpretation is a mapping that takes an input structure Σ in a signature $\mathcal{S}$ and uses logical expressions to recast it as an output structure Γ in a signature $\mathcal{G}$, shown abstractly in Figure 2. One way to imagine this is interpreting an output structure Γ *through the lens of an input structure* Σ. It is also convenient to imagine this as a transformation, where an input structure is transformed into an output structure. Here, the term *logical transduction* is used.

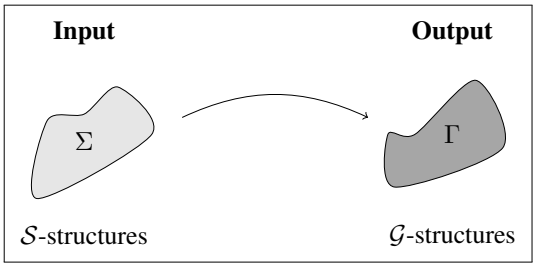

Figure 2: General Sketch of a Logical Transduction

Let $\mathcal{G}$ be our output signature with relations $r_1', \ldots, r_n'$. For each relation r_i' in the output signature $\mathcal{G}$, there must be a definition which is defined using only relations r_i from the input signature $\mathcal{S}$ or more complex helper predicates constructed from them. There is also a copyset $C = \{1, \ldots, m\}$ that copies pieces of the input structure to be (potentially) used by the output structure. Essentially for each node x in the input structure Σ, depending on the size of the copyset, a corresponding copy is used: there can be an x^0 copy, x^1

copy, x^2 copy, and so on up to m, meaning that the input structure will grow linearly depending on the size of the copyset.

Consider the input signature $\mathcal{S}$ (from the string model for *abpa* in Figure 1) and the output signature $\mathcal{G}$, containing precedence relations that mediate precedence between copies:

$$\mathcal{S} = \{a(x), b(x), p(x), \lhd(x,y)\}$$
$$\mathcal{G} = \{a^0(x), p^0(x), b^0(x), a^1(x), p^1(x), b^1(x), \lhd^{0,0}(x,y), \lhd^{0,1}(x,y), \lhd^{1,0}(x,y), \lhd^{1,1}(x,y)\}$$

In the output signature $\mathcal{G}$, the relations $\sigma^0(x)$ mean that x's 0-th copy is labeled with the symbol σ and $\sigma^1(x)$ mean that x's 1-st copy is labeled with the symbol σ. The relations $\lhd^{i,j}(x,y)$ mediate strict precedence *between different copies* in the output. In other words, $\lhd^{i,j}(x,y)$ means that x's i-th copy strictly precedes y's j-th copy in the output. This is made clear in the example that follows.

Using these two signatures, we will construct an input $\mathcal{S}$-structure Σ, an output $\mathcal{G}$-structure Γ, and an interpretation will be constructed between them that epenthesizes a's between a p followed by a b. This can be written as a standard rewrite rule $\varnothing \rightarrow a / p\_b$. Note that a copyset of $C = \{0, 1\}$ is needed because the string will grow in length by one node any time there is a 'pb' substring. We proceed by defining each relation in the output signature using relations from the input signature. The interpretation is shown pictorially in Figure 3.

For the labeling relations, every node in the 0-th copy is going to remain faithful to the input. Nothing is deleted, there are only things to add and so this copy remains the same. In the 1-st copy, only nodes labelled with an a will ever appear since that is the only segment we wish to add (since b's or p's will never be epenthesized). For the precedence relations, strict precedence will hold between two nodes in the 0-th copy if they aren't a p strictly followed by a b. The only time a 0-th copy will strictly precede a 1-st copy is when there is an a being inserted, namely between a p strictly followed by a b. The only time a 1-st copy will precede a 0-th copy is when it is the b in the configuration just described. There will never be strict precedence between elements both in the 1-st copy, since this would correspond to adding two a's in a row.

In Figure 3, the dashed nodes represent copies of nodes that are not used in the interpretation. When the copyset is constructed, all copies *have the potential to be used*, but the actual definitions of the labeling and precedence relations determine which are actually used. In this mapping, an input string $apba$ will map to $apaba$, since the a was epenthesized between the p and b, whereas an input string $abapa$ would simply map to $abapa$ since there are no 'pb' substrings.

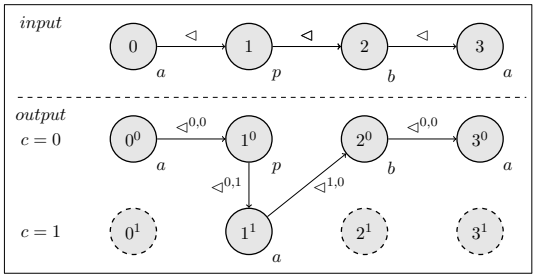

Figure 3: a-epenthesis between p and b

Thinking more generally, this is an example of how an output string has a particular form *based on specified conditions on its corresponding input string*. Thus, we are interpreting the output string through the lens of the input string. Shifting focus to linearization, an output string structure has a particular form based on specified conditions on its corresponding input tree structure. To understand this more clearly, tree models must first be defined.

3 Linearization as an Interpretation

3.1 Tree Models

It is standard practice within model-theoretic syntax to define trees with respect to a domain $D \subsetneq \mathbb{N}$ of nodes, a binary general dominance relation $\lhd^*(x,y)$ and a left-of/precedence relation $\prec (x,y)$ as in Rogers and Nordlinger (1998). There are many theoretical reasons to suggest that they should instead be defined over something more closely resembling syntactic selection instead of a precedence relation, but in order to keep the discussion more tractable, this convention serves as a suitable starting point. Some ways that this can be embellished for a more well-rounded account will be discussed in later sections.

Note that the domain ranges over the natural numbers $\mathbb{N}$, but the order that they appear doesn't matter so long as the relations are consistently defined. For convenience, the convention here reflects the order that they are introduced to the derivation, assuming a bottom up derivation.[1]

[1]One could also choose to use Gorn addresses as in Lam-

There must also be labels for the nodes of our tree, so let Σ_{syn} be an input alphabet of labels for nodes of our tree. Since our nodes can bear a wide range of different syntactic properties, this alphabet can be partitioned into the following sets of lexical labels, categories, features, and movement-licensing features:

- $L = \{\text{THE}, \text{MAN}, \text{LOVES}, \text{CAKE}, \dots\}$
- $C = \{\text{V}, \text{V}, \text{C}, \text{D}, \text{N}, \text{PERF}, \dots\}$
- $F = \{\text{SG}, \text{PL}, 1, \dots\}$
- $LIC = \{\texttt{+wh}, \texttt{-wh}, \texttt{+nom}, \texttt{-nom}, \dots\}$

Thus, $\Sigma_{syn} = L \cup C \cup F \cup LIC$, and each $\sigma \in \Sigma_{syn}$ has a corresponding unary relation $\sigma(x)$ specifying some piece of syntactic information. This is one particular choice of how to encode this information relationally, inspired by Minimalist Grammars (Stabler, 1996), but many other options are available. While this is not strictly necessarily, a labelless syntax is assumed (Collins, 2002), such that non-terminal nodes without *lexical* labels (bearing no $\sigma \in L$) represent instantiations of Merge.[2]

We will start with a simplified, abstract example for clarity and it will be expanded when movement is discussed. Consider the input signature:

$$\Sigma = \{\lhd^*(x, y), \prec (x, y), \sigma_i(x)\}$$

where:

- $\lhd^*(x, y)$ is the binary general dominance relation
- $\prec (x, y)$ is the binary, asymmetric precedence relation
- $\sigma_i(x)$ are unary relations for every $\sigma_i \in \Sigma_{syn}$

To keep the discussion tractable while introducing the main properties of linearization, only lexical labels are encoded in this structure, but the general points about how labels carry over to the output structure hold for the other category and feature labels. A simplified example of an $\mathcal{S}$-structure in the signature Σ is shown in Figure 4 over the arbitrary, abstract alphabet $\Sigma_{syn} = \{\text{THE}, \text{MAN}, \text{LOVES}, \text{THE}, \text{CAKE}\}$.

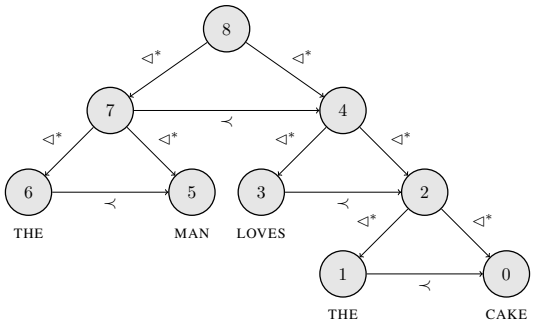

Figure 4: Linearization Toy Example

Considering this example, setting aside the issue of movement for later, the linearization intuitively yields the string "THE MAN LOVES THE CAKE". However, this is a nontrivial task since branches can be of arbitrary finite length. The next section lays out an interpretation using First-Order Logic to yield a string defined using strict precedence.

3.2 Remarks on Linearization

The main contribution of this work is to show that linearization can be concisely understood as an interpretation between trees and strings. In order to formalize this, it is crucial to first establish some theoretical assumptions of both input trees and output strings.

There is a rich body of work debating the status of linearity and recursion and their presence in syntax and phonology (Scheer, 2012, 2023; Idsardi and Raimy, 2013; Idsardi, 2018; Elfner, 2015; Ito and Mester, 2012; Cheng and Downing, 2021; Miller and Sande, 2021). This paper adheres to the view that (i) *narrow syntax contains recursion but lacks linearity* and (ii) *phonology contains linearity but lacks recursion*. To understand this, looking at work by Idsardi and Raimy (2013) is helpful. They outline three types of linearization, one of which, *immobilization*, plays a key role here. Immobilization transforms hierarchical structures built via Merge into ordered structures by introducing adjacency relations. There is a subtle but crucial point here with respect to the status of linearity in the computation of narrow syntax. The structure building taking place during narrow syntactic computation is blind to linearity, but linearity is a necessary reflex of externalization given

bert et al. (2021), where domain elements are strings in $\{0, 1\}^*$ where a 0 indicates a left child and a 1 indicates a right child.

[2] A series of well-formedness conditions can be defined that more accurately reflect standard syntactic assumptions (the nodes that select project, encoding feature percolation, etc.), some of which will be explored later with respect to movement.

the temporal nature of the speech stream. So there must be a stage after syntactic structures are built which imposes linearity, and this is precisely the function of immobilization. The finer details of immobilization are beyond the scope of this paper, but similar model-theoretic tools are well-suited to formalize it. In this framework, we assume that "flattening" occurs after adjacency relations are established. Thus, the input trees of our linearization are the recursive hierarchical trees built by the narrow syntax *once they have been embellished with adjacency information*, hence the use of the precedence relation $\prec (x, y)$.

In what follows, we define this mapping using First-Order Logic, ensuring that the process remains sufficiently restrictive from a computational perspective. This formulation allows linearization to be expressed in a purely declarative manner rather than as a derivational process. It also fundamentally codifies the notion of locality into the representation, which is known to be important for the post-syntax (Embick and Noyer, 1999).

3.3 Tree-Flattening as an Interpretation

Consider an input $\mathcal{S}$-structure, a tree denoted Σ, and an output string $\mathcal{G}$-structure, a string denoted Γ, representing the concatenation of Σ's leaves in the correct order. Recall that the relations of our output string must be defined in terms of those input relations (namely, $\lhd^*, \prec, \sigma_i$ or helper predicates built using these) and this is precisely the sense in which the output string is being interpreted in terms of the input tree.

As before, two pieces are necessary: (i) which nodes from the input are relevant for the output and (ii) how they are ordered with respect to each other. The ordering will be a relation called $\texttt{lin}(x, y)$ to indicate that x and y in the input tree meet the conditions for x to strictly precede y in the linearized output string. Intuitively, only the leaves will be contained in the output structure, but the ordering between them may not be readily clear at first glance. Taking the tree in Figure 4, its intended linearization shown pictorially below in Figure 5. The example will proceed by reasoning why the ordering is the way it is, which will lead to the formal definition.

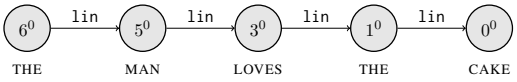

Figure 5: Output of Linearization Toy Example

We only want to include leaf nodes in our output string, and because the input will not grow in the output, we only need a single copy set $C = \{0\}$. In fact, this interpretation can be seen as a mapping that "forgets" the hierarchical information and "connects" the leaves in the correct order via linear precedence. We define a predicate $\texttt{leaf}(x) := \neg\exists y[\lhd^*(x, y)]$ that says a node x is a leaf node iff there is no node y that it dominates. Thus, the labeling relations will take the following form for each item in the input alphabet $\sigma_i \in \Sigma_{syn}$:

$$\text{THE}^0(x) := \text{THE}(x) \wedge \texttt{leaf}(x)$$
$$\text{MAN}^0(x) := \text{MAN}(x) \wedge \texttt{leaf}(x)$$
$$\vdots$$

To better understand why the output string has the linear order it does, some more helper predicates are defined. A `left-leaf` is a leaf that has nothing preceding it, and a `right-leaf` is a leaf that precedes nothing. Formally,

$$\texttt{left-leaf}(x) := \texttt{leaf}(x) \wedge \neg\exists y[\prec (y, x)]$$
$$\texttt{right-leaf}(x) := \texttt{leaf}(x) \wedge \neg\exists y[\prec (x, y)]$$

Using these, we can define predicates to indicate whether a given node is the *left-most leaf* of a particular node, and another to indicate if a given node is the *right-most leaf* of a particular node. For a given node, whichever node is the (unique!) leaf below it such that nothing is further left is its left-most leaf and whichever node is the (unique!) leaf below it such nothing is further right is its right-most leaf.

The relevance of these becomes clear when thinking about where $\texttt{lin}(x, y)$ holds true in the tree in Figure 4. Let's observe each case: First, $\texttt{lin}(6, 5)$ because both 6 and 5 are leaves and $\prec (6, 5)$. Next, $\texttt{lin}(5, 3)$ because there is a node whose right-most leaf is 5 and it precedes a node whose left-most leaf is 3, so no other leaves can be in between them. Next, $\texttt{lin}(3, 1)$ because 3 precedes a node whose left-most leaf is 1. Finally, $\texttt{lin}(1, 0)$ for the same reason $\texttt{lin}(6, 5)$, namely both are leaves and $\prec (1, 0)$.

Thus, in all of these scenarios, expressing strict precedence in the output requires reference to left-most and right-most leafhood. Every node has a left-most and right-most leaf, and every leaf node is its own right-most and left-most leaf (since dominance is taken to be reflexive). Defining one more

helper predicates aids in readability. The following predicate indicates that a node y is dominated by x and dominates z and so we say that y is *between* x and z in the tree:

$$\texttt{between}(x, y, z) := \lhd^*(x, y) \wedge \lhd^*(y, z)$$

Now having seen the importance of these configurational relationships to linearization, the formal definitions for right-most and left-most leafhood are as follows:

- A node x is the left-most leaf of a node y iff for all the left-leaf nodes z that y dominates, the only one with nothing further left is x:

$$\begin{aligned}\texttt{lml}(x, y) := \forall z[(\lhd^*(y, z) \wedge \texttt{left-leaf}(z) \\ \wedge \forall s[\texttt{between}(y, s, z) \\ \wedge \neg\exists t[\prec (t, s)]]) \leftrightarrow z = x]\end{aligned}$$

- A node x is the right-most leaf of a node y iff for all the right-leaf nodes z that y dominates, the only one with nothing further right is x:

$$\begin{aligned}\texttt{rml}(x, y) := \forall z[(\lhd^*(y, z) \wedge \texttt{right-leaf}(z) \\ \wedge \forall s[\texttt{between}(y, s, z) \\ \wedge \neg\exists t[\prec (s, t)]]) \leftrightarrow z = x]\end{aligned}$$

Now that these have been given, note that each of the cases above made some mention of x and y being the left-most or right-most leaf of two higher nodes where one precedes the other, we can call these t and s.[3] Any of these configurations leading to x strictly preceding y in the output string can be condensed into the following single condition, also shown pictorially in Figure 6:

$$\texttt{lin}(x, y) := \exists t \exists s[\prec (t, s) \wedge \texttt{rml}(t, x) \wedge \texttt{lml}(s, y)]$$

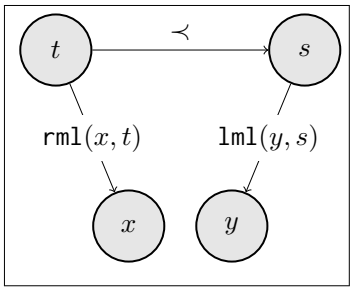

Figure 6: Conditions for Strict Precedence in Output

[3]Since any leaf is its own left-most leaf and right-most, it can be true that either $t = x$ or $s = y$ or both.

What we have done is reduced precedence between any two nodes in the output string to a *single declarative condition* between nodes the input tree: the node x will strictly precede y iff this condition holds. One of the primary strengths of this result is that it doesn't cast linearization in terms of a procedure, but rather it reduces it to underlying knowledge about the structural relationship between linguistic elements. Another critical property of this method of linearization is that it is definable using First-Order Logic, which is desirable from a formal complexity standpoint. This is because it limits its use of quantification to individual elements as opposed to sets of elements as would be the case in Monadic Second Order Logic. This is a nice result with respect to computational complexity, since it is subregular.

There is an important question regarding the choice of strict precedence in the output string. Recall from the earlier discussion of strings that the choice of representation (strict or general precedence) affects the available generalizations one can define. Using strict precedence it is natural to ban immediately adjacent segments like a ban on any obstruent immediately following a nasal (say, a *NT constraint). For example, with an alphabet of $\Sigma = \{\text{V}, \text{N}, \text{T}, \text{D}\}$, such a constraint would accept the string VNDV but reject the string *VNTV. In fact, this is a strictly local constraint since it depends only on a window of two elements (Chandlee, 2014). In contrast, using general precedence it is natural to ban sequences of segments like long distance sibilant harmony that bans an ∫ following an s anywhere in the string (say, a *s ... ∫ constraint). For example, with an alphabet of $\Sigma = \{\text{ʃ},\text{s},\text{v},\text{c}\}$, such a constraint would accept the string ʃcvcvʃ but reject the string *scvcvʃ. This is not strictly local because it contains a dependency between elements that can occur arbitrarily far away from one another. [4] The choice of strict precedence in the definition of linearization here formally hard-codes locality into our post-syntactic representations. The output of our linearization is a string of nodes labeled with morphosyntactic information and if they are related via strict precedence, this prunes out arbitrary, word-level parallels of these long distance generalizations. Thus, post-syntactic operations at

[4]However, it is possible to define Tier-Based Input or Output Strictly Local functions that have a relativized form of adjacency via a particular feature or category, thus naturally constraining the ability to make long-distance generalizations.

this level of representation can be modeled using ISL functions.

This simplified example did not contain any feautural information, but this will become relevant when sketching a potential analysis implementing movement. As a start, encoding standard syntactic mechanisms like selectional requirements and feature percolation can be stated as well-formedness conditions on our input trees. As an example, suppose we had a well-formedness condition in our trees that said a non-terminal node only bears a category label, for example D, iff it has two children x, y where x shares the category D and $\prec (x, y)$, which enforces that the selecting node will project its features to its parent. Another example, suppose we define a well-formedness condition for movement features `f` which states that if a leaf node bears a `-f` feature, this `-f` feature must percolate upward to its maximal projection. These are some ways to understand how this method of linearization could be expanded going forward for a more all-encompassing account.

3.4 Incorporating Movement

There are many ways one could imagine incorporating movement to this analysis. One potential way is to assume that we have a tree-to-tree mapping, where the input tree is a pre-movement tree and the output is a post-movement tree. While this does split the division of labor, a notable drawback of this approach is that it would require two separate interpretations: one solely for completing movement and another for linearization. There is also the question of how to encode movers. This could be done by embellishing the alphabet with movement traces, where our trees would instead have trace labels at the launching sites and lexical labels at their landing sites. This would drastically increase the length of the alphabet since this would presumably require a trace corresponding to each label already in L.

Another potential alternative would entail altering some of our representational assumptions for input trees. Our input trees could be modified to include a separate relation to encode movement. For example, suppose we had a relation $\mathcal{M}(x, y)$ where x is the the highest node of a mover and y is a node immediately dominating a movement attracting head. This could then be used to define a structural input condition to determine the placement of x's children in the output string.

The alternative sketched here assumes movement takes place concurrently with linearization, sketched using an example in Figure 7. While an analysis in which trees are built with a syntactic selection relation as opposed to precedence may reflect the nature of syntactic computation more accurately, this would be beyond the scope of this paper. Incorporating the exhaustive well-formedness conditions, movement configurations, successive cyclic movement or enforcing relativized minimality in all generality would be considerably much more involved than is possible here, but such an analysis is left for further work. Given the fact that most work in model-theoretic syntax has assumed the sorts of representations used here, this is sufficient for the central points regarding linearization.

In the tree in Figure 7, substructures that consist of movers are darkened for clarity. In the output string, the string yield of the movers is outlined with a dashed box to clarify that these are the leaves of an *entire substructure* with relevant properties from the input. There are two moving substructures in this tree. One is the substructure with the root 9, the phrase "THE MAN", driven by a `nom` feature and the other is the substructure with the root 2, the phrase "WHICH CAKE", driven by a `wh` feature. In the output string, the moved phrase driven by `nom` appears to the left of the attracting head T bearing a `+nom` feature. Similarly, the moved phrase driven by `wh` appears to the left of the attracting head C bearing a `+wh` feature bearing a `-wh` feature.

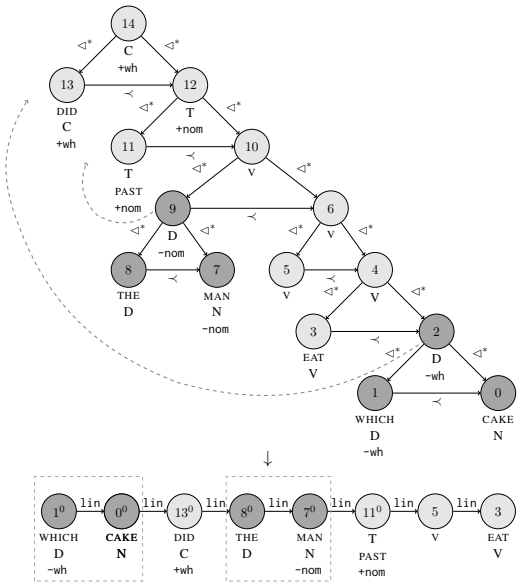

Figure 7: Linearization Toy Example with Movement

What is true about each of these moving substructures with respect to these heads? Each of them are rooted with a node that bears a `-f` feature for some movement-driving feature `f`, as per the well-formedness condition posited earlier. Thus, the yield of this substructure should occur before the `f`-movement driving head in the output string, meaning precisely that the `rml` of the moving substructure will strictly precede this `+f` head. In the case of the `nom` movement, the output string structure will have 7 (the MAN-bearing node) strictly preceding 11 (the `+nom`-bearing T node). Similarly, in the case of the `wh` movement, the output string structure will have 0 (the CAKE-bearing node) strictly preceding 13 (the `+wh`-bearing C node). Together with the $\texttt{lin}(x, y)$ condition, this covers the movers themselves and their relationship to the movement-driving heads.

There are two remaining tasks: we must determine what precedes the mover once it lands and the nodes around its launching site. Firstly, it must be ensured that the mover's `lml` comes before the unique node which would have met $\texttt{lin}(x, y)$ in the input (where y is the movement attracting head). For example, C will strictly precede the `nom`-mover's `lml` bearing THE. Similarly, the node bearing WHICH will be the first node in the string since there is nothing higher than the attracting head. Secondly, it must be ensured that the nodes surrounding the mover, if they exist, are connected via strict precedence. In other words, the next highest node that would have met $\texttt{lin}(x, y)$ where y is the `lml` of the mover should strictly precede the next lowest node that would have met $\texttt{lin}(x, y)$ where x is the `rml` of the mover. For example, the node bearing T will strictly precede the node bearing V since they "surround" the launching site. Similarly, the node bearing EAT will be the final node in the string since there is nothing lower than the mover in the input tree.

This is only sketched out as an example, but the entirety of the mapping just described is definable by making modifications to the $\texttt{lin}(x, y)$ condition within First-Order Logic. This is because in what was just described, it only requires quantification of individual nodes, not arbitrary subsets of nodes, leaving the definition within First-Order Logic. Even though this substructure can be arbitrarily large, the only relevant nodes of the mover to be picked out are its root, `rml` and `lml` and nodes in between are covered by $\texttt{lin}(x, y)$. An anonymous reviewer points out that the scope here is limited to a relatively simple case of movement, but further work could provide a more structured analysis of more complex cases (e.g. multiple movers attracting to a single head, smuggling, remnant movement, mixed-headedness, etc.) using these tools and examine whether they remain within First-Order Logic.

4 Discussion

This novel view of linearization comes with many theoretical advantages. Firstly, it was shown (albeit through automata-theoretic as opposed to model-theoretic means) that Recursive Prosody is non finite state and thus requires more computational power (Dolatian et al., 2021). This declarative *linearization-as-flattening* approach has the benefit that it only uses First-Order Logic, which is notably less computationally expensive than Monadic Second-Order Logic, which is required for mappings that are finite state or more powerful.

This approach may have interesting implications for our view of the syntax-phonology interface regarding the status of recursion in phonology. For a recent view on the debate of the status of recursive prosodic approaches and procedural approaches, see Lee and Selkirk (2022); Newell and Sailor (in press). It is well-known that there are often mismatches between syntactic and phonological domains (Cheng and Downing, 2016); however, these mismatches often appear to occur at or very near to Spell-out boundaries. Accommodating this notion of "at or very near to" is extremely amenable to this type of analysis given its inherent locality properties. If string yields are embellished with boundary information (either by means of boundary symbol like $\rtimes$ or $\ltimes$, respectively or relations that hold of a node $\varphi_{\texttt{init}}(x)$ or $\varphi_{\texttt{fin}}(x)$, respectively), then it may be expected that the range of syntax-phonology mismatches are accounted for through by employing Input Strictly Local (ISL) restructuring functions (Chandlee, 2014), which are very computationally restrictive. Dobashi (2003, 2019) has work detailing phonological domain restructuring and its typological implications. These approaches are nicely compatible and would serve as a fruitful integration of theoretical and computational results at the syntax-phonology interface, creating a new avenue for more formal analyses in this domain.

There are other computational characterizations of linearization that exist currently; for example, (Graf, 2022a,b) gives an elegant formal characterization which is ISL, a strikingly desirable property with respect to computational complexity; however, this does require abiding by quite strong representational assumptions about the nature of trees that are undoubtedly formally well-founded and rigorous, but have not received a wider adoption in more general syntactic literature. The analysis makes use of dependency trees, which are relatively uncommon outside of the space of computational syntax. While Graf defines a straightforward mapping between more standard phrase structure trees and dependency trees, the analysis proposed here takes a view where linearization occurs straight from more standard syntactic representations dispensing with the need for such intermediate mappings. This also adds to a recent body of work that has begun to bridge the gap between theoretical work on the interface and separate computational work in phonology and syntax (Dolatian et al., 2021; Yu, 2021; Vu et al., 2022; Stabler and Yu, 2023), despite some of the differing theoretical assumptions regarding the status of recursion.

5 Conclusion

This paper has presented a novel method of linearization, casting it as a model-theoretic interpretation between strings and trees. It is both computationally restrictive and hard-codes locality into the output string representations, all while expressing ordering between nodes as a single declarative condition. A potential expansion incorporating movement was explored through a motivating example, showing that the tools are amenable to further modifications. The most central advantage to this analysis is the fact that it is a step toward computationally unifying how we think about linguistic modules and their interaction, despite some of their representational differences.

Acknowledgements

I would like to thank Adam Jardine and the attendees of the 2024 Rutgers Subregular Phonology Workshop for their valuable insights and discussion on this work. I would also like to thank the three anonymous reviewers for their thoughtful and constructive feedback.

References

Siddharth Bhaskar, Jane Chandlee, Adam Jardine, and Christopher Oakden. 2020. Boolean monadic recursive schemes as a logical characterization of the subsequential functions. In *Language and Automata Theory and Applications: 14th International Conference, LATA 2020, Milan, Italy, March 4-6, 2020, Proceedings 14*, pages 157–169. Springer.

Jane Chandlee. 2014. *Strictly local phonological processes*. University of Delaware.

Lisa Lai-Shen Cheng and Laura J Downing. 2016. Phasal syntax= cyclic phonology? *Syntax*, 19(2):156–191.

Lisa Lai-Shen Cheng and Laura J Downing. 2021. Recursion and the definition of universal prosodic categories. *Languages*, 6(3):125.

Chris Collins. 2002. Eliminating labels. *Derivation and explanation in the Minimalist Program*, pages 42–64.

Yoshihito Dobashi. 2003. *Phonological phrasing and syntactic derivation*. Cornell University.

Yoshihito Dobashi. 2019. *Externalization: Phonological interpretations of syntactic objects*. Routledge.

Hossep Dolatian, Aniello De Santo, and Thomas Graf. 2021. Recursive prosody is not finite-state. In *Proceedings of the Seventeenth SIGMORPHON Workshop on Computational Research*.

Emily Elfner. 2015. Recursion in prosodic phrasing: Evidence from connemara irish. *Natural Language & Linguistic Theory*, 33:1169–1208.

David Embick and Rolf Noyer. 1999. Locality in post-syntactic operations. *MIT working papers in linguistics*, 34(265-317).

Thomas Graf. 2022a. Diving deeper into subregular syntax. *Theoretical Linguistics*, 48(3-4):245–278.

Thomas Graf. 2022b. Subregular linguistics: bridging theoretical linguistics and formal grammar. *Theoretical Linguistics*, 48(3-4):145–184.

Jeffrey Heinz. 2018. *The computational nature of phonological generalizations*, pages 126–195. De Gruyter Mouton, Berlin, Boston.

William Idsardi and Eric Raimy. 2013. Three types of linearization and the temporal aspects of speech. *Challenges to linearization*, 1:31–56.

William J. Idsardi. 2018. *Why Is Phonology Different? No Recursion*, pages 212–223. Cambridge University Press.

Junko Ito and Armin Mester. 2012. Recursive prosodic phrasing in japanese. *Prosody matters: Essays in honor of Elisabeth Selkirk*, pages 280–303.

Adam Jardine. 2017. On the logical complexity of autosegmental representations. In *Proceedings of the 15th Meeting on the Mathematics of Language*, pages 22–35.

Dakotah Lambert, Jonathan Rawski, and Jeffrey Heinz. 2021. Typology emerges from simplicity in representations and learning. *Journal of Language Modelling*, 9.

Seunghun J. Lee and Elisabeth Selkirk. 2022. Xitsonga tone: The syntax-phonology interface. In *Prosody and Prosodic Interfaces*. Oxford University Press.

Taylor L Miller and Hannah Sande. 2021. Is word-level recursion actually recursion? *Languages*, 6(2):100.

Scott Nelson. 2024. *The Computational Structure of Phonological and Phonetic Knowledge*. Ph.D. thesis, State University of New York at Stony Brook.

Heather Newell and Craig Sailor. in press. *Minimalism and the Syntax-Phonology Interface*. Cambridge University Press.

Christopher Donal Oakden. 2021. *Modeling phonological interactions using recursive schemes*. Ph.D. thesis, Rutgers The State University of New Jersey, School of Graduate Studies.

Andrija Petrovic. 2023. *Insights From the Interfaces: Morphological Processes as String Transductions*. Ph.D. thesis, State University of New York at Stony Brook.

Jonathan Rawski. 2021. *Structure and Learning in Natural Language*. Ph.D. thesis, State University of New York at Stony Brook.

James Rogers and Rachel Nordlinger. 1998. *A descriptive approach to language-theoretic complexity*. Citeseer.

Tobias Scheer. 2012. Chunk definition in phonology: prosodic constituency vs. phase structure. *Modules and interfaces*, pages 221–253.

Tobias Scheer. 2023. Recursion in phonology: Anatomy of a misunderstanding. *Representing phonological detail: Segmental structure and representations*, pages 265–287.

Edward Stabler. 1996. Derivational minimalism. In *International conference on logical aspects of computational linguistics*, pages 68–95. Springer.

Edward P Stabler and Kristine M Yu. 2023. Unbounded recursion in two dimensions, where syntax and prosody meet. *Proceedings of the Society for Computation in Linguistics*, 6(1):343–356.

Kristina Strother-Garcia, Jeffrey Heinz, and Hyun Jin Hwangbo. 2016. Using model theory for grammatical inference: a case study from phonology. In *Proceedings of The 13th International Conference on Grammatical Inference*, volume 57 of *JMLR: Workshop and Conference Proceedings*, pages 66–78.

Mai Ha Vu, Aniello De Santo, and Hossep Dolatian. 2022. Logical transductions for the typology of ditransitive prosody. In *Proceedings of the 19th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 29–38.

Kristine M Yu. 2021. Computational perspectives on phonological constituency and recursion. *Catalan journal of linguistics*, 20:77–114.