Sensing Tree Automata as a Model of Syntactic Dependencies

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

Aniello De Santo
Department of Linguistics
Stony Brook University
Stony Brook, NY, USA
aniello.desanto@stonybrook.edu

Abstract

Various aspects of syntax have recently been characterized in subregular terms. However, these characterizations operate over very different representations, including string encodings of c-command relations as well as tiers projected from derivation trees. We present a way to unify these approaches via sensing tree automata over Minimalist grammar dependency trees. Sensing tree automata are deterministic top-down tree automata that may inspect the labels of all daughter nodes before assigning them specific states. It is already known that these automata cannot correctly enforce all movement dependencies in Minimalist grammars, but we show that this result no longer holds if one takes into account several well-established empirical restrictions on movement. Sensing tree automata thus furnish a strong yet uniform upper bound on the complexity of syntactic dependencies.

1 Introduction

This paper proposes a novel, unified upper bound on the subregular complexity of syntactic dependencies. Since the merit of this result might be opaque to a reader who is not intimately familiar with the most recent developments in subregular complexity, we start out with a very detailed background discussion.

The subregular program seeks to identify formal machinery that provides a tighter characterization of natural language than the familiar classes of the Chomsky hierarchy. Subregular phonology took as its vantage point the well-known result that phonology is regular (Johnson, 1972; Kaplan and Kay, 1994) and then identified proper subclasses that are still powerful enough for specific types of phonological dependencies. Among these subclasses are SL, SP (Rogers et al., 2010), IBSP (Graf, 2017, 2018a), TSL (Heinz et al., 2011; McMullin, 2016), and several extensions of the latter (Baek, 2018; Graf and Mayer, 2018). The highly restrictive nature of these classes has allowed for new learning algorithms (Heinz et al., 2012; Jardine and McMullin, 2017) and also furnishes computational explanations for typological gaps.

Syntax cannot be subregular in this strict sense by virtue of being at least mildly context-sensitive (Huybregts, 1984; Shieber, 1985; Michaelis and Kracht, 1997; Kobele, 2006, a.o.). However, linguists formulate syntactic dependencies over trees, not strings. This shift in perspective has also taken place within formal grammar, first with model-theoretic syntax (Blackburn et al. 1993; Backofen et al. 1995; Cornell and Rogers 1998; Rogers 1998, 2003, a.o.) and then the two-step approach (see Morawietz 2003, Mönnich 2006, and references therein). The two-step approach highlighted how the mildly context-sensitive nature of syntax arises from the interaction of two finite-state components: a regular tree language that encodes a kind of "deep structure" and a finite-state tree transduction to the intended "surface structure". This perspective has proven particularly fruitful in Minimalist grammars (MGs; Stabler, 1997, 2011a), where derivation trees provide the regular tree language and the transduction is a formal analogue to Chomsky's notion of movement (Kobele et al., 2007; Graf, 2012b). With the regular nature of syntax properly identified, a subregular characterization of syntax is suddenly feasible.

Graf (2012a) provided the first subregular (un)definability results for MG derivation tree languages, but these results were expanded on only recently. Notably, all follow-up work strived to remain closer to the classes that enjoy prominence in subregular phonology. This, however, also led to a marked divergence in approaches. Graf (2018b) operates directly over MG derivation trees. Following the tradition of model-theoretic syntax, Graf equates the MG operations Merge and Move

with constraints on MG derivation tree languages and shows that they belong to the tree analogue of the subregular string class TSL. This view is also adopted by Vu (2018) and Vu et al. (2019) in the analysis of negative-polarity items and case licensing, respectively. Graf and Shafiei (2019) and Shafiei and Graf (2019), on the other hand, pursue a purely string-based perspective of syntactic dependencies. For each node they identify its string of c-commanders, the shape of which must follow the constraints imposed by, say, Principle A or NPI-licensing. When construed as such string constraints, syntactic licensing conditions not only turn out to be subregular, they also fit into classes that have been proposed for subregular phonology. The work so far thus has unearthed two distinct subregularity results: MG operations are subregular over MG derivation trees, and licensing conditions are subregular over a specific string representation grounded in c-command (cf. Frank and Vijay-Shanker, 2001).

Even though each perspective is worthwhile and has proven very fruitful, their apparent incommensurability raises the question how these two notions of subregularity can be brought to bear on each other. The central contribution of our paper is a uniform upper bound on syntax that encompasses both MG operations and licensing conditions. This upper bound takes the form of sensing tree automata (STAs) operating over dependency tree representations of MG derivations (these dependency trees are distinct from the MG dependency trees of Boston et al. 2010). STAs provide a minimal amount of look-ahead to deterministic top-down tree automata: the automaton may inspect the labels of all daughter nodes before assigning them specific states. Far more than just a mathematical curiosity, limiting syntax to STA-recognizable constraints over MG dependency trees is very natural in several respects:

1. MG dependency trees are a natural encoding of head-argument relations.

2. STAs explain why licensing conditions are mediated by c-command instead of the many alternative command relations one could imagine (Barker and Pullum, 1990).

3. In order for movement to be regulated by STAs, it must obey additional restrictions beyond those of the standard MG formalism. These restrictions coincide with well-known empirical phenomena such as the Specifier Island Constraint and the Coordinate Structure Constraint.

4. As a single STA can handle movement and licensing conditions at the same time, it is unsurprising that the two occasionally interact, e.g. when movement induces a licensing configuration or violates one.

5. Since STAs are deterministic top-down automata with a minimal amount of look-ahead, they are a natural match for top-down parsing, which has been argued to play a central role in human sentence processing (Stabler, 2013; Kobele et al., 2013; Graf et al., 2017).

Hence our contribution does not merely unify the two existing approaches to subregular syntax, it also accounts for empirical aspects of syntax that the latter leave unexplained. At the same time, we do not intend for our perspective to supplant the existing ones. Each one provides useful insights and can now be safely pursued in the knowledge that there is a principled formal connection to the other approaches.

The paper is laid out as follows: Section 2 introduces our mathematical notation for trees and tree languages (§2.1), which form the basis for our definition of STAs (§2.2). We then define MGs and their dependency trees in §3. In order to simplify some of the subsequent proofs, we do not follow the standard definitions and instead adopt a format that is partly inspired by Kobele et al. (2007). Sections 4 and 5 cover the two core results of this paper: dependency trees of MGs with the Specifier Island Constraint are STA-recognizable, and so are licensing conditions based on c-command. The last section briefly sketches some extensions of these results, including interactions of movement and c-command (§6.1), subcommand (§6.2), recognizability of MG derivation trees (§6.3), adjunct islands and the coordinate structure constraint (§6.4, §6.5), and connections to top-down parsing (§6.6).

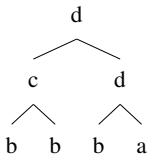
2 Preliminaries

2.1 Trees and Tree Languages

A *ranked alphabet* Σ is a finite set of symbols, each one of which has a *rank* or *arity* 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, the set $T(\Sigma)$ of Σ-*trees* contains all $\sigma^{(0)}$ and all terms $\sigma^{(n)}(t_1, \ldots, t_n)$ $(n \geq 0)$ such that $t_1, \ldots, t_n \in T(\Sigma)$.

Example 1. Given $\Sigma := \{a^{(0)}, b^{(0)}, c^{(2)}, d^{(2)}\}$, $T(\Sigma)$ is an infinite sets that contains, among others, the term $d(c(b, b), d(b, a))$. This term corresponds to the tree below:



If Σ consisted only of $c^{(2)}$ and $d^{(2)}$, then $T(\Sigma)$ would be empty. ⌟

Given a term $m^{(n)}(s_1, \ldots, s_n)$ where each s_i is a subtree with root d_i, we call m the *mother* of the *daughters* $d_1, \ldots, d_n$ $(1 \leq i \leq n)$. If two distinct nodes have the same mother, they are *siblings*. We use the term *proper dominance* for the transitive closure of the mother-of relation, and *reflexive dominance* for the reflexive, transitive closure. A node a is an *ancestor* of node n iff a properly dominates n.

Every node u in a Σ-tree has a unique address $g(u)$ as defined in Gorn (1967). The subtree of Σ-tree t rooted in Gorn address $g(u)$ is denoted by $t/g(u)$. Given two Σ-trees s and t and node u in t, $t[g(u) \leftarrow s]$ is the result of replacing $t/g(u)$ in t by s. Throughout the paper, we write u instead of $g(u)$ so that t/u and $t[u \leftarrow s]$ are shorthands for $t/g(u)$ and $t[g(u) \leftarrow s]$, respectively.

2.2 Sensing Tree Automata

A *sensing tree automaton* (STA; Martens et al., 2008) is a deterministic top-down tree automaton that may also take the labels of a node's daughters into account before assigning states to them. In other words, these automata have a finite look-ahead of 1. This intuition is reflected in our definition of *sensing rules* below. Following the format of Comon et al. (2008, p. 38) for top-down tree automata, we deliberately define sensing rules in a tree transducer format. This is a marked deviation from Martens et al. (2008), but it should make it easier for future work to extend our perspective from constraints to transformations.

Let Σ be some ranked alphabet and Q a set of symbols of rank 1 disjoint from Σ. Members of Q are called *states*. In addition, X is a countably infinite set of variable symbols ($X \cap \Sigma = X \cap Q = \emptyset$). For the rest of this section, we use $\vec{\sigma_i}$ as a shorthand for $\sigma_i(x_{i_1}, \ldots, x_{i_n})$, with $\sigma_i \in \Sigma^{(n)}$ and $x_{i_1}, \ldots, x_{i_n} \in X$.

Definition 1 (Sensing rule). A *sensing rule* over alphabet Σ and state set Q is an expression of the form $q(\sigma^{(n)}(\vec{\sigma_1}, \ldots, \vec{\sigma_n})) \to \sigma^{(n)}(q_1(\vec{\sigma_1}), \ldots, q_n(\vec{\sigma_n}))$ such that $\sigma, \sigma_1, \ldots, \sigma_n \in \Sigma$ and $q, q_1, \ldots, q_n \in Q$. ⌟

Note that for $\sigma^{(0)}$, sensing rules are of the form $q(\sigma^{(0)}) \to \sigma^{(0)}$. Such sensing rules effectively remove states from leaf nodes, whereas sensing rules for $\sigma^{(\geq 1)}$ remove the state of σ and instead add a state on top of each daughter of σ.

Definition 2 (Sensing relation). Let t and t' be in $T(\Sigma \cup Q)$, and suppose δ is some sensing rule over Σ and Q that has the form $q(\sigma^{(n)}(\vec{\sigma_1}, \ldots, \vec{\sigma_n})) \to \sigma^{(n)}(q_1(\vec{\sigma_1}), \ldots, q_n(\vec{\sigma_n}))$. Then the relation $\to_\delta$ holds between t and t' ($t \to_\delta t'$) iff t contains a subtree $s := q(\sigma^{(n)}(s_1, \ldots, s_n))$ such that each s_i $(1 \leq i \leq n)$ is a Σ-tree with root σ_i, and t' is the result of replacing s in t with $\sigma^{(n)}(q_1(s_1), \ldots, q_n(s_n))$.

Given a set Δ of sensing rules, we write $t \to_\Delta t'$ iff $t \to_\delta t'$ for some $\delta \in \Delta$. The reflexive, transitive closure of $\to_\Delta$ is denoted by $\to^*_\Delta$. ⌟

Intuitively, $t \to^*_\Delta t'$ iff t' can be obtained from t by a sequence of sensing rules.

Definition 3 (Sensing tree automaton). A *sensing tree automaton* (STA) over Σ is a 4-tuple $A := \langle Q, \Sigma, q_I, \Delta \rangle$ such that Q is a finite set of states disjoint from alphabet Σ, $q_I \in Q$ is the initial state, and Δ is a finite set of *sensing rules* over Σ and Q. The tree language recognized by A is $L(A) := \{t \in T(\Sigma) \mid q_I(t) \to^*_\Delta t\}$. The class of all tree languages that are recognized by at least one STA is called STA. ⌟

An STA thus recognizes a tree t iff there is a sequence of sensing rules that starts from the initial state q_I and passes states through t until they are all removed again at the leaves of t. If the STA ever gets stuck because there is no suitable sensing rule, t is rejected.

Example 2. Suppose $\Sigma := \{a^{(0)}, b^{(0)}, c^{(2)}, d^{(2)}\}$, and consider the STA with $Q := \{0, 1\}$, where 0 is the initial state. The automaton uses all sensing rules of the following form:

- $1(\sigma(\vec{\sigma_1}, \vec{\sigma_2})) \to \sigma(1(\vec{\sigma_1}), 1(\vec{\sigma_2}))$,
  for $\sigma \in \{c, d\}$ and $\sigma_1, \sigma_2 \in \{a, b, c, d\}$,

- $0(\sigma(\vec{\sigma_1}, \vec{\sigma_2})) \to \sigma(0(\vec{\sigma_1}), 0(\vec{\sigma_2}))$,
  for $\sigma \in \{c, d\}$ and $\sigma_1, \sigma_2 \in \{a, b, d\}$,
- $0(\sigma(\vec{\sigma_1}, \vec{\sigma_2})) \to \sigma(q_0(\vec{\sigma_1}), q_1(\vec{\sigma_2}))$,
  for $\sigma \in \{c, d\}$, $\sigma_1, \sigma_2 \in \{a, b, c, d\}$, and $q_i := 1$ iff $\sigma_{2-i} := c$ (where $i \in \{0, 1\}$),
- $q(b) \to b$,
  for $q \in \{0, 1\}$,
- $1(a) \to a$.

The STA only accepts those Σ-trees where each a is c-commanded by some c (that is to say, c must be the sibling of an ancestor of a). It thus emulates a simplified version of Principle A. ⌟

It will also be convenient in some proofs to employ a substitution-based characterization of the tree languages that are recognized by STAs. The characterization capitalizes on the fact that every STA only has a look-ahead of 1. Consequently, the state it assigns to a node n depends only on three components: I) the label of n and its siblings, II) the states assigned to n's ancestors, and III) the states assigned to the siblings of each ancestor. If all of those are kept constant, n will always receive the same state no matter what the rest of the tree looks like.

Definition 4 (Spine closure). Given a node u of some Σ-tree t, $\text{lsib}^t(u)$ is the string consisting of the label of u's left siblings (if they exist) followed by the label of u. Analogously, $\text{rsib}^t(u)$ is the string consisting of the label of u and the label of its right siblings (if they exist). Also, $\bigtriangledown$ and $\Downarrow$ are two distinguished symbols not in Σ.[1] Let $u_1, \ldots, u_n$ be the shortest path of nodes extending from the root of t to u. That is to say, each u_i is the mother of u_{i+1} ($1 \leq i < n$), u_1 is the root, and $u_n = u$. By $\text{spine}^t(u)$ we denote the string recursively defined by $\text{spine}^t(u_1) = u_1 \bigtriangledown u_1$ and $\text{spine}^t(u_1, \ldots, u_n) = \text{spine}^t(u_1, \ldots, u_{n-1}) \Downarrow \text{lsib}^t(u_n) \bigtriangledown \text{rsib}^t(u_n)$ A regular tree language L is *spine-closed* iff it holds for all trees $s, t \in L$ and nodes u and v belonging to s and t, respectively, that $\text{spine}^s(u) = \text{spine}^t(v)$ implies $s[u \leftarrow t/v] \in L$. ⌟

Theorem 1 (Martens 2006). *A regular tree language L belongs to the class* STA *iff L is spine-closed.*

[1] Martens et al. (2008) use # instead of $\Downarrow$. We prefer the latter as it emphasizes visually that this symbol marks the start of a string at the next lower level in the tree.

Example 3. Consider the left tree l and right tree r in Fig. 1. Let $l0$ and $r0$ be the left daughter of the root in l and r, respectively. Then $\text{spine}^l(l0) = \text{spine}^r(r0) = \mathit{merge} \bigtriangledown \mathit{merge} \Downarrow \mathit{merge} \bigtriangledown \mathit{merge}\ \mathit{merge}$. Hence a tree language that contains both l and r is an STA language iff it also contains $r[r0 \leftarrow r/l0]$. ⌟

In linguistic terms, spine-closure tells us that two subtrees s/u and t/v with identical root labels can be freely exchanged whenever they have the same ancestors and the same c-commanders. This will be of great importance throughout this paper.

3 Minimalist Grammars

Minimalist grammars (MGs; Stabler, 1997) are a formalization of Minimalist syntax (Chomsky, 1995). Readers who are unfamiliar with the formalism should consult Stabler (2011a) for a more accessible introduction.

Every MG consists of a finite set of feature annotated lexical items. Each lexical item is a pair of a phonetic exponent and a finite, non-empty string of features. There are four types of features, whose job it is to trigger the structure-building operations *Merge* and *Move*. Merge establishes head-argument relations and is triggered when a *selector feature* F^+ on a head finds a matching *category feature* F^- on an argument. For example, the noun *guest* carries a feature N^-, for which we also write guest :: N^-. It can be merged with the :: N^+D^- to yield a DP, thanks to the matching category and selector features. Move displaces a subtree from its current position to a higher position in the syntactic structure. Move takes place when a *licensor features* f^+ on a head that provides a landing site can be checked by a corresponding *licensee feature* f^- on a mover. The order of features on a lexical item determines the order in which the corresponding operations are triggered. Hence the determiner which :: $\text{N}^+\text{D}^-\text{wh}^-$ would first select a noun phrase, get merged with a head looking for a DP, and then undergo wh-movement.

The sequence of Merge and Move steps is commonly represented as a derivation tree, e.g. in Fig. 2. However, we will use a dependency tree representation instead. Our dependency trees are merely a more compact encoding of MG derivation trees and have no connection to the MG dependency trees of Boston et al. (2010). We will

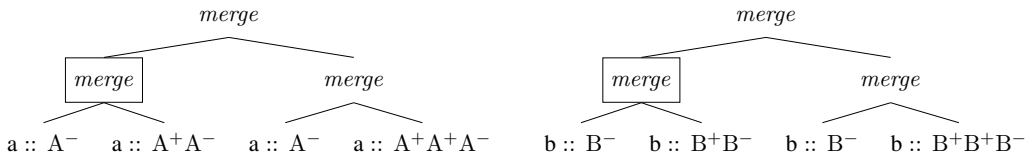


Figure 1: MG derivation tree languages are not recognizable by STAs because they are not spine-closed.

directly define MGs as sets of well-formed dependency trees, mirroring earlier definitions in terms of derivation trees (primarily Kobele et al. 2007).

We pick a ranked alphabet Σ such that $\Sigma^{(n)}$ contains all lexical items of MG G, and only those, that carry exactly n selector features. For simplicity, we use G to also refer to this alphabet. Not every G-tree is a well-formed dependency tree, though, due to the constraints of the MG feature calculus. The calculus is illustrated in Fig. 2, where each node in the dependency tree is annotated with the feature configuration corresponding to its subtree. We formalize this calculus via a recursive function feat that computes these values based on more primitive functions for the feature checking steps that trigger Merge and Move.

Definition 5 (Dependency tree language). If G is an MG with n distinct licensee features, then the set $\mathrm{dep}(G)$ of well-formed dependency trees of G is $\{t \in T(G) \mid \mathrm{feat}(t) = \langle \mathrm{C}^-, \varepsilon_1, \ldots, \varepsilon_n \rangle\}$. ⌟

The remainder of this section defines feat in terms of feature checking operators M and $\otimes$ for Move and Merge, respectively. All operations manipulate one or more sequences of feature strings. The Shortest Move Constraint (SMC) will be used to filter out illicit sequences.

Definition 6 (SMC). Let G be an MG and Lce its set of n licensee features. Given a sequence $s := s_1, \ldots, s_m$ ($m \geq 0$) of strings in Lce*, $\mathrm{SMC} : (\mathrm{Lce}^*)^* \to (\mathrm{Lce}^*)^*$ is undefined for s if there are s_i and s_j ($1 \leq i \neq j \leq m$) that start with the same licensee feature. Otherwise, SMC maps s to s itself. ⌟

The SMC ensures that Move is unambiguous in the sense that there can never be more than one active mover of a specific type (wh, topicalization, and so on). It is an integral part of MGs, and removing it would greatly alter their expressivity (Salvati, 2011).

Next we add a helper function sort that orders SMC-approved sequences based on the first licensee feature of each feature string.

Definition 7 (sort). Let G and Lce be as before. Now fix some bijection b between Lce and $\{1, \ldots, n\}$. Then sort maps $s := s_1, \ldots, s_m \in \mathrm{Lce}^*$ to the sequence $s'_1, \ldots, s'_n$ such that $s'_i := s_j$ if s_j starts with f and $b(f) = i$; otherwise, $s'_i := \varepsilon$ ($1 \leq i \leq n, 1 \leq j \leq m$). ⌟

Now we can finally define M for Move, which is also a crucial part of the Merge operator $\otimes$. Throughout we use γ as a shorthand for any string of features, and δ for a (possibly empty) string of licensee features.

Definition 8 (Move). Suppose that expression e is $\langle f\gamma, \delta_1, \ldots, \mathrm{f}^-\delta_i, \ldots, \delta_n \rangle$ for some licensee feature f^- and $1 \leq i \leq n$. Then

$$M(e) := M(\langle \gamma, \mathrm{sort}(\mathrm{SMC}(\delta_1, \ldots, \delta_i, \ldots, \delta_n)) \rangle)$$

if f is the licensor feature f^+, and e otherwise. ⌟

Definition 9 (Merge). Given two expressions $e := \langle f\gamma, \delta_1, \ldots, \delta_m \rangle$ and $e' := \langle f'\delta, \delta_{m+1}, \ldots, \delta_z \rangle$, $e \otimes e'$ is

$$M(\langle \gamma, \mathrm{sort}(\mathrm{SMC}(\delta, \delta_1, \ldots, \delta_m, \delta_{m+1}, \ldots, \delta_z)) \rangle)$$

if f is some selector feature F^+ and f' the matching category feature F^-. In all other cases, $\otimes$ is undefined. ⌟

Note how Merge is always followed by an application of the Move operator, but this does not trigger any feature checking unless γ starts with a licensor feature. Merge steps are thus interleaved with movement checks, not all of which may actually result in movement.

The operators M and $\otimes$ on their own do not narrow down the set of G-trees. They are invoked as part of a recursive function feat over G-trees that computes the feature values of subtrees, as already expressed in Def. 5.

Definition 10 (feat). The partial function feat recursively maps MG dependency trees to feature expressions. For lexical items, $\mathrm{feat}(\sigma :: \phi) := \langle \phi, \varepsilon_1, \ldots, \varepsilon_n \rangle$. If $t := \sigma(\sigma_1, \ldots, \sigma_z)$, then $\mathrm{feat}(t) := ((\mathrm{feat}(\sigma) \otimes \mathrm{feat}(\sigma_z)) \otimes \cdots) \otimes \mathrm{feat}(\sigma_1)$. ⌟

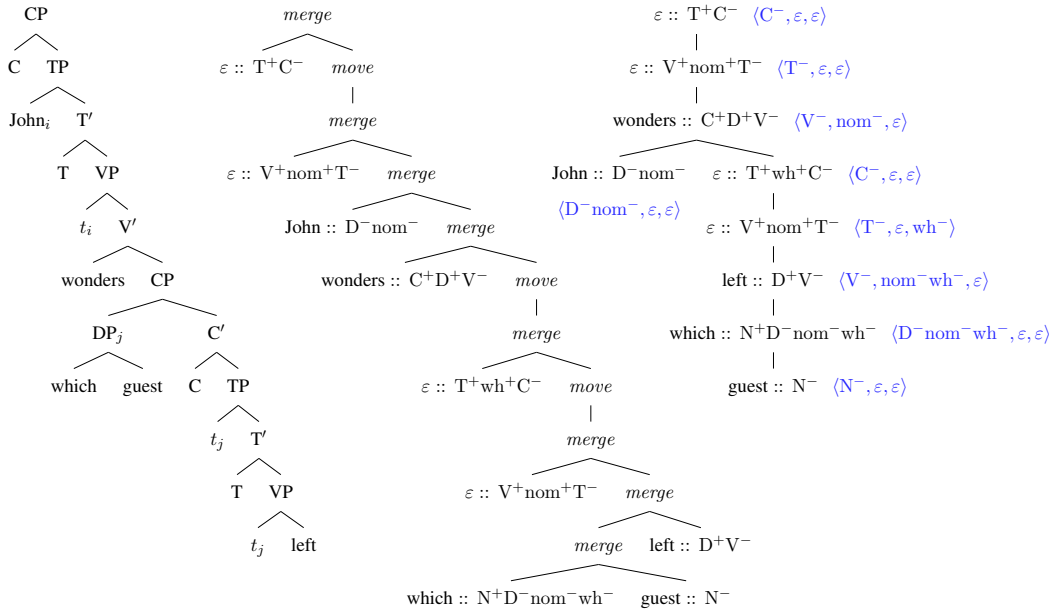


Figure 2: X′-bar tree, corresponding MG derivation tree, and (feat-annotated) MG dependency tree

Two important lemmata follow immediately from the preceding definitions.

Lemma 1. *Let G be an MG and s a G-tree. Then it holds for every $t \in \mathrm{dep}(G)$ with node u that $t[u \leftarrow s] \in \mathrm{dep}(G)$ iff $\mathrm{feat}(s) = \mathrm{feat}(t/u)$.*

Lemma 2. *Let G be an MG with n distinct licensee features and s a subtree of some $t \in \mathrm{dep}(G)$. Then $\mathrm{feat}(s)$ must be of the form $\langle \mathrm{F}^-\delta, \delta_1, \ldots, \delta_n\rangle$ for some category feature F^-.*

Lemma 2 is apparent from the derivation tree example in Fig. 2. It is a minor extension of the well-known fact that a lexical item may occur in a well-formed MG derivation iff its feature string is of the form $\phi F \delta$, where ϕ is either ε or a selector feature followed by 0 or more selector and licensor features, F is a category feature, and δ is a (possibly empty) string of licensee features.

4 Merge and Move via STAs

With all the preliminaries in place, we can finally turn to the core results regarding the STA-recognizability of MGs with respect to Merge and Move. The next sections then extend this to licensing conditions and some other special cases.

Graf (2012a) uses the argument from example 3 in §2.2 to prove that MG derivation tree languages are not STA languages. However, this proof does not carry over to MG dependency trees. Adopting the terminology of Graf (2012a), we use $\textsc{mdep}[merge, move]$ for the full class of MG dependency tree languages and $\textsc{mdep}[merge]$ for the subclass of movement-free MGs (no lexical item carries any licensee features).

Theorem 2. $\textsc{mdep}[merge] \subsetneq \textsc{sta}$

This is just a corollary of a more fundamental property of MGs. For any arbitrary $L \in \textsc{mdep}[merge]$ and nodes u and v of $s, t \in L$, $\mathrm{spine}^s(u) = \mathrm{spine}^t(v)$ necessarily entails that $\mathrm{feat}(s/u) = \mathrm{feat}(t/v)$, so that Theorem 2 immediately follows from Lemma 1. We omit a full proof here as Lemma 3 will cover a more complex case that subsumes this one.

Even with the dependency tree format, though, STAs are too weak for standard MGs with both Merge and Move.

Theorem 3. $\textsc{mdep}[merge, move]$ *and* STA *are incomparable.*

Proof. Consider the dependency trees l and r in Fig. 3. The respective instances of the :: N$^+$D$^-$ have different values under feat ($\langle \mathrm{D}^-, \varepsilon\rangle$ and $\langle \mathrm{D}^-, \mathrm{wh}^-\rangle$, respectively). By Lemma 1, then, their subtrees are not interchangeable even though $\mathrm{spine}^l(\text{the :: N}^+\mathrm{D}^-) = \mathrm{spine}^r(\text{the :: N}^+\mathrm{D}^-)$. □

The example in Fig. 3 is peculiar, though. The left dependency tree encodes the derivation for *Who does a teacher of like the father of John*, which is severely degraded. It has been argued that such cases of left-branch subextraction are generally forbidden. MGs can be equipped with the *Specifier Island Constraint* (SpIC) to

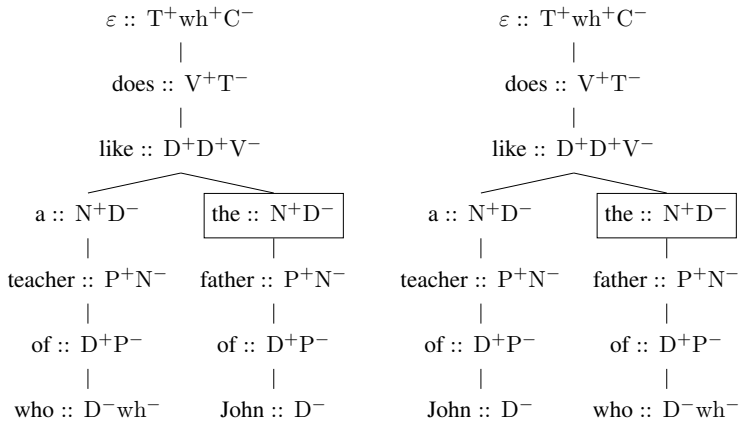


Figure 3: Even though the boxed nodes have the same spine, their subtrees cannot be exchanged.

rule out movement from within a specifier. This takes the form of an additional restriction on feat that only allows complements to properly contain unchecked licensee features. Since the complement of a head is its rightmost daughter in the dependency tree (rather than the leftmost one), the SpIC amounts to a restriction on all daughters except the last one.

Definition 11 (SpIC). Suppose $s_1, \dots, s_m$ are G-trees and $\sigma \in G^{(m)}$. Then $\text{feat}(\sigma(s_1, \dots, s_m))$ is undefined if there is an $i < m$ and $1 \leq j \leq n$ such that $\text{feat}(s_i)$ is of the form $\langle \gamma, \delta_1, \dots, \delta_j, \dots, \delta_n \rangle$ and $\delta_j \neq \varepsilon$.

Note that any licensee features on the head of a specifier are part of γ, not δ_j, so specifiers can still move without violating the SpIC. Only extraction of a proper subtree from within a specifier is not allowed. Also note that our version of the SpIC only bans extraction from base specifiers, but not from specifiers that are derived via movement. This is why it can be easily stated over dependency trees. Even so, this limited version of the SpIC greatly limits the weak generative capacity of MGs with the SMC, while still keeping them mildly context-sensitive (Michaelis, 2004, 2009; Kobele and Michaelis, 2011). For our purposes, though, the major contribution of the SpIC is that it also lowers the complexity of MG dependency trees into STA.

Lemma 3. *Given an MG G that obeys the SpIC, pick arbitrary nodes u and v of $s, t \in \text{dep}(G)$, respectively. If $\text{spine}^s(u) = \text{spine}^t(v)$, then $\text{feat}(s/u) = \text{feat}(t/v)$.*

Proof. Since s and t are well-formed, both $\text{feat}(s/u)$ and $\text{feat}(t/v)$ must be defined. By Lemma 2 we may assume w.l.o.g. that $\text{feat}(s/u) := \langle \text{F}^-\delta, \delta_1, \dots, \delta_n \rangle$ and $\text{feat}(t/v) := \langle \text{F}'^-\delta', \delta'_1, \dots, \delta'_n \rangle$. As u and v have identical spines, u and v themselves must be identical. Therefore both $\text{F}^- = \text{F}'^-$ and $\delta = \delta'$ hold. Now suppose that $\delta_i \neq \delta'_i$ for some $1 \leq i \leq n$. Then either s/u or t/v is missing a licensee feature f^- that is present in the other. Suppose it is s/u that is missing a feature present in t/v. Note that this immediately entails by the SpIC that t/v is a complement, wherefore s/u is also a complement because u and v have identical spines. Since both s and t are well-formed, whatever feature is missing in s/u must occur somewhere else in s to match some f^+ in the spine of u. But by the SpIC, f^- cannot occur properly inside any specifier. We already know that s/u is a complement, so f^- can only occur on an ancestor of u or on one of its left siblings. But then it would occur on a node in the spine of u, which contradicts our initial assumption that $\text{spine}^s(u) = \text{spine}^t(v)$. Hence $\delta_i = \delta'_i$ after all, wherefore $\text{feat}(s/u) = \text{feat}(t/v)$. □

Theorem 4. *For every MG G that obeys the SpIC, it holds that $\text{dep}(G) \in$ STA.*

Proof. Lemma 1 and 3 jointly imply that $\text{dep}(G)$ is spine-closed, which guarantees that it can be recognized by an STA (Thm. 1). □

Intuitively, Theorem 4 holds because the SpIC creates a unique "elsewhere case" for missing movers. Suppose that an STA is at node n in a G-tree t. Since it has processed t top-down, it knows exactly which movers it has to look for by virtue of the licensor features it has come across. With its look-ahead of 1, the STA can scan the daughters of n to see if any of them carry some of the desired licensee features. Any licensee features that

I can show this as an example of why the SpIC makes it SDA definable

are not among them must be embedded deeper in the tree. Due to the SpIC, though, they can only reside in the complement, i.e. the subtree rooted in the rightmost daughter of n.

The SpIC is just one way of creating such an elsewhere case. STA-recognizability would also hold if movers could only escape from the leftmost argument, or if the label of the selecting head decides which one of its arguments can be extracted from. Extraction from arguments could also be parameterized for each feature so that wh-movers may only leave complements whereas topicalization is only allowed from the last but one argument, if it exists. Or the STA could switch between these four constraints depending on the number of ancestors of the current node *modulo* 4. STA-recognizability holds as long as distributing the head's δ_i across its arguments is fully deterministic based on the information available to a sensing rule (current state, label of selecting head, labels of selected heads). Hence the class of STA-recognizable MG dependency tree languages is larger than what is allowed by the SpiC.

This does not change the fact, though, that the initial finding of Graf (2012a) regarding the insufficiency of STAs is incomplete. In the case of Merge, the insufficiency disappears with the more compact representation format of MG dependency trees. Alternatively, one could also keep the derivation tree format while increasing the STA look-ahead beyond just one level — this is a point we will revisit soon in §6.2 and §6.3, for very different reasons. With respect to Move, the choice of representation format is immaterial. Neither derivation trees nor dependency trees make movement as defined in MGs STA-recognizable. However, the SpIC does make movement STA-recognizable because specifiers do not need to be probed deeper than their head. For MG dependency trees, this coincides with the 1-level look-ahead of STAs, whereas derivation trees once again require a more generous look-ahead window. The choice of representation thus has an impact on the amount of required look-ahead, but the essence of our STA-recognizability result for MGs rests on the SpIC, not the tree format.

5 STAs and C-Command Conditions

The previous section has successfully established that the central operations of MGs can be handled by STAs, assuming that I) they are construed as constraints on MG dependency trees, and II) movement is subject to the SpIC. But the structure-building operations Merge and Move are just one part of syntax. Licensing conditions also play a major role, in particular those rooted in c-command. This section shows that these conditions are also captured by STAs.

Licensing conditions based on c-command are ubiquitous in the syntactic literature. They were recently studied from a subregular perspective by Graf and Shafiei (2019) and Shafiei and Graf (2019). Both papers use similar ideas, but define them very differently. We adopt the formalism of Graf and Shafiei (2019) because it defines all essential concepts directly in terms of dependency trees.

Definition 12 (C-string). Let t be some MG dependency tree. For every node n of t in configuration $m(d_1, \ldots, d_i, n, d_{i+1}, \ldots, d_j)$, its *immediate c[ommand]-string* is $\mathrm{ics}(n) = d_1 \cdots d_i\ n$. The *augmented c[ommand]-string* $\mathrm{acs}(n)$ of n is recursively defined as shown below, where $\uparrow$ is a distinguished symbol:

$$\mathrm{acs}(n) := \begin{cases} \mathrm{ics}(n) & \text{if } n \text{ is the root of } t \\ \mathrm{acs}(m) \uparrow \mathrm{ics}(n) & \text{if } m \text{ is } n\text{'s mother} \end{cases}$$

⌟

Example 4. The c-string of the :: $\mathrm{N^+D^-}$ in Fig. 3 is ε :: $\mathrm{T^+wh^+C^-} \uparrow$ does :: $\mathrm{V^+T^-} \uparrow$ like :: $\mathrm{D^+D^+V^-} \uparrow$ a :: $\mathrm{N^+D^-}$ the :: $\mathrm{N^+D^-}$ ⌟

Licensing conditions are then formalized as constraints on the shape of permissible c-strings. This is comparable to restricting a tree via its path language, except that paths are now replaced by c-strings.

Definition 13 (C-string constraints). A c-string constraint C is some string language L over $\Sigma \cup \{\uparrow\}$. A Σ-tree t is well-formed with respect to C iff $\mathrm{acs}(n) \in L$ for every node n of t. ⌟

Which subregular class provides the most appropriate fit for syntactic licensing conditions is still a matter of debate. Graf and Shafiei (2019) propose IO-TSL as a generous upper bound. IO-TSL is a subregular string language recently defined in (Graf and Mayer, 2018) in their analysis of Sanskrit n-retroflxion. Shafiei and Graf (2019), on the other hand, argue that at least island constraints are best captured by IBSP, another class from subregular work on phonology (Graf, 2017, 2018a). Neither class is particularly well-understood at this

point. Intuitively, IO-TSL treats local dependencies as primitive and reduces non-local constraints to local ones over enriched representations. IBSP, on the other hand, takes all constraints to be non-local and then uses locality domain to prune down their reach. Either way there is ample evidence that all attested conditions that can be correctly stated over c-strings are at most regular. That's all we need to show that they can be enforced by an STA.

Before we proceed, the reader should take note that c-strings as defined in Graf and Shafiei (2019) do not quite capture the standard notion of c-command over phrase structure trees. First of all, movement is factored out, so that c-strings only capture the c-command relations between the base positions where arguments enter the derivation. Graf and Shafiei (2019) point out several options for adding movement, but they do not explore them in depth. We will provide our own STA account for movement interactions later on (§6.1). Another, less important deviation from standard c-command pertains to the status of heads and specifiers. If one interprets linear precedence in command strings as c-command, then specifiers do not c-command their selecting head even though they c-command the head's object. At the same time, the head c-commands the specifier. While there seem to be no cases in the syntactic literature where this difference to c-command matters, it nonetheless highlights that c-strings only approximate the standard notion of c-command.

This approximation of c-command is very convenient for our purposes, though. The construction of a node's c-string closely mirrors the definition of a node's spine, so it is perhaps unsurprising that every regular c-string constraint can be enforced by an STA. The construction of an STA automaton for this purpose is remarkably straightforward.

First, we simplify c-string constraints by considering only constraints that generate prefix-closed sets of c-strings. The lemma below establishes that this assumption is innocuous for determining tree well-formedness.

Lemma 4. *Let L be some regular language of well-formed c-strings, and let L_p be the largest subset of L such that $u \notin L_p$ entails $uv \notin L_p$ for all $u \in (\Sigma \cup \{\uparrow\})^+$ and $v \in (\Sigma \cup \{\uparrow\})^*$. Then a Σ-tree is well-formed with respect to L iff it is well-formed with respect to L_p.*

Proof. We only consider c-strings that do not start or end with $\uparrow$ as these never occur in any trees to begin with. Since $L_p \subseteq L$ and, by Def. 13, a tree is well-formed with respect to c-string set C iff all its c-strings are members of C, two entailments follow immediately: I) if t is well-formed with respect to L_p, it is well-formed with respect to L, and II) if t is ill-formed with respect to L, it is ill-formed with respect to L_p.

Next, suppose t is well-formed with respect to L. Since it contains no illicit c-string, t could only be ill-formed with respect to L_p if it contains some licit c-string uv such that $u \notin L_p$, wherefore $uv \notin L_p$. But if uv is a c-string of t, then so is every non-empty prefix of uv that does not end in $\uparrow$. By our initial assumption, t is well-formed, and hence every non-empty prefix of uv is a member of L. It then must also be a member of L_p because, by definition, L_p must be largest among the prefix-closed subsets of L. It follows that t is well-formed with respect to L_p, too.

Finally, consider the case where t is ill-formed with respect to L_p. Then there is some c-string u of t such that $u \notin L_p$ but every non-empty prefix of u (that does not end in $\uparrow$) is a member of L_p. In this case it must also hold that $u \notin L$, for otherwise L_p is either not largest or violates prefix closure. Consequently, t is also ill-formed with respect to L. □

Intuitively, Lemma 4 capitalizes on the fact that whenever a tree contains at least one unlicensed node, the status of other nodes no longer matters because the tree is already ill-formed. Hence we may freely assume that the c-string of node n is illicit as soon as a prefix of that c-string is illicit, even in cases where n itself would be licensed.

Now let $D := \langle \Sigma \cup \{\uparrow\}, Q, q_I, F, \delta \rangle$ be the complete, deterministic finite-state string automaton that recognizes L_p as defined in Lemma 4. As D is deterministic, it has a unique initial state q_I. Its transition relation $\delta : (Q \times \Sigma) \times Q$ is a function. We expand δ to strings such that $\delta(q, \sigma_1\sigma_2 \cdots \sigma_n) := \delta(\cdots \delta(\delta(q, \sigma_1), \sigma_2) \cdots, \sigma_n)$. Since L_p is prefix-closed, it also holds that $Q = F \cup \{s\}$, where $s \notin F$ is some sink state from which no other state can be reached except s itself. Prefix closure also entails that $q_I \in F$, so that empty c-strings are always allowed (since by definition c-strings are never empty, this is innocuous).

We then construct the corresponding STA

$A_D := \langle \Sigma, Q, q_I, \Delta \rangle$. For $n \geq 1$, each sensing rule in Δ is of the form

$$\begin{aligned} q(\sigma^{(n)}(\vec{\sigma_1}, \vec{\sigma_2}, \ldots, \vec{\sigma_n})) \rightarrow \\ \sigma(\delta(q, \sigma\uparrow)(\vec{\sigma_1}), \\ \delta(q, \sigma \uparrow \sigma_1)(\vec{\sigma_2}), \\ \ldots, \\ \delta(q, \sigma \uparrow \sigma_1\sigma_2 \cdots \sigma_{n-1})(\vec{\sigma_n}))) \end{aligned}$$

where $q \in F$ and, as previously defined at the beginning of §2.2, the use of $\vec{\sigma_i}$ with some symbol $\sigma_i^{(n)}$ is a shorthand for $\sigma_i(x_{i_1}, \ldots, x_{i_n})$. For leaf nodes, we require $q(\sigma^{(0)}) \rightarrow \sigma^{(0)} \in \Delta$ iff both $q \in F$ and $\delta(q, \sigma) \in F$.

This construction effectively simulates runs of the string automaton D over c-strings. The only complication is that the process of assigning a state to σ_i does not consider σ_i itself. But σ_i does affect the states of its right siblings and all its daughters. And if σ_i is a leaf, it indirectly determines whether the state can be removed so that the tree may be accepted.

Theorem 5. *Let L be a regular language of well-formed c-strings. Then there is some STA A such that $L(A)$ is the set of all Σ-trees that are well-formed with respect to L.*

Proof. Following Lemma 4, we replace L by L_p and consider the complete, deterministic automaton D that generates L_p. We construct A from D in the manner described above. We then give a proof by induction on the depth of Σ-trees.

Pick some Σ-tree t and suppose $t \in \Sigma^{(0)}$. The only c-string of t is t. Then A recognizes t iff $q_I \in F$ and $\delta(q_I, t) \in F$. The former holds by definition, so $t \in L(A)$ iff $t \in L(D)$. This establishes the base case.

Next, consider any arbitrary configuration $q(\sigma^{(n)}(\sigma_1, \ldots, \sigma_n))$. By our induction assumption, $q = \delta(q_I, u)$ with $\mathrm{acs}(\sigma) = u\sigma$. Suppose $\mathrm{acs}(\sigma_i) \notin L_p$ for some $1 \leq i \leq n$. Then $\delta(q_I, \mathrm{acs}(\sigma_i)) = \delta(q, \sigma \uparrow \sigma_1 \cdots \sigma_i) \notin F$. If $i < n$, A will assign some non-final state to σ_{i+1}. If $i = n$ and σ_i is not a leaf, A assigns the non-final state to the leftmost daughter of σ_i. In both cases, t now contains some node with a non-final state. But there is no sensing rule with a non-final state on its left-hand side, so that A correctly rejects t. If $i = n$ and σ_i is a leaf, then $\delta(\delta(q, \sigma \uparrow \sigma_1 \cdots \sigma_{i-1}), \sigma_i)$ is not final and consequently there is no suitable leaf rewrite rule. As this covers all possible configurations for σ_i, we conclude that A rejects every tree that is ill-formed with respect to L_p.

In the other direction, suppose A rejects t. Then there must be some configuration $\sigma(q_1(\sigma_1), \ldots, q_i(\sigma_i), \ldots, q_n(\sigma_n))$ such that q_i is not a final state. But then $\mathrm{acs}(\sigma_{i-1}) \notin L_p$, as desired. □

Theorem 5 establishes that the same subregular machinery of STAs can be used for both the structure-building operations Merge and Move and the c-command licensing conditions that deeply permeate syntax. The STA perspective also provides a new answer as to why c-command should play such an important role in syntax. How an STA treats a given node depends solely on the spine of that node. The spine contains the node itself, all its ancestors, and the siblings of all the nodes in the spine. This immediately precludes generalized notion of command like the S-command relation of Barker and Pullum (1990), which in modern terminology would be CP-command: x CP-commands y iff x does not reflexively dominate y (or the other way round) and every CP that properly dominates x properly dominates y. As x and y can be arbitrarily deep within distinct subtrees while S-commanding each other, this is not an STA-recognizable command relation. C-command, on the other hand, stays within the narrow confines of STAs.

Admittedly STAs could also selectively ignore some c-commanders, operate with "inverse c-command" where a complement c-commands into its specifiers, or switch between different notions of c-command based on some *modulo* counting condition. So just as with the SpIC, the power of STAs goes quite a bit beyond what is desirable for c-command. Still, it is striking that c-command is a very natural relation from STA perspective, whereas more global notions of command are correctly ruled out. As long as one is willing to accept dependency structures as a natural representation that arises from head-argument relations, c-command is a natural companion of STA-recognizability.

A lot of work remains to be done, though. As just discussed, STAs allow for some very unnatural command relations. Even more troubling is that any arbitrary regular constraint over c-strings can be enforced by an STA. Seeing how syntactic constraints are very limited in the shape of depen-

dencies they enforce, STAs are overly powerful. Hence STAs can only act as an upper bound.

At the same time, our current STA approach is too limited. In cases where licensing conditions interact with movement, the licensing element might not be part of the c-string of a node. Our STA construction, which is based purely on c-strings, will necessarily fail in these cases. In addition, some cases of licensing involve generalized notions of c-command that go beyond what STAs can handle. We are confident, though, that these issues can be addressed in future work. The next section briefly sketches the solutions we have in mind.

6 Expanding the Core Results

6.1 Interactions of Movement and Licensing

Linguists have identified many cases where movement obfuscates licensing configurations by displacing the licensed element. A simple example would be *[which book about himself]$_i$ does John like* t_i, where the reflexive is not c-commanded by its antecedent *John* in the corresponding phrase structure tree. These cases are entirely unproblematic for STAs since the MG dependency trees fully factor out movement, so *which book about himself* remains in the object position where it is c-commanded by the subject *John*. The problematic cases are much rarer, to such a degree that convincing examples are hard to come by: I) movement of a licensed element bleeding licensing, and II) movement of a licensing element feeding licensing.

The first case covers configurations where a licensed element — or a subtree containing it — moves to a position above the element's licensor and where the subsequent change in c-command relations does make licensing impossible (in contrast to the binding example above, where licensing holds nonetheless). In MGs with the SpIC this can be captured by an STA. The states of the STA would be n-tuples similar to the output of the feat-function we defined for MGs. The first component records c-string states in the usual fashion, whereas each other component i records the most recent head h with an f_i^+-feature, plus the state that was assigned to h. When an f_i-mover m is found, the information in the i-th component is used to compute the state for m as if m were a left daughter of h. In this case, m is also excluded from the state computation of its siblings in the dependency tree. The details remain to be worked out, but movement of a licensee should be easy enough to handle because the STA can separately keep track of the c-command configurations for each position that is targeted by a mover.

The second option is more complicated. Here some phrase must move into a higher position from where it can c-command the element that must be licensed. In combination with the SpIC, this means that the licensing of a node can be contingent on the nodes contained by its rightmost sibling. An STA can still handle this, but it requires a very different construction from the one described in this paper. As in the bleeding case, a state consists of multiple components, each one of which corresponds to a movement feature. Component i records all the types of licensing conditions that still must be met for nodes that are c-commanded from the most recent head with licensor feature f_i^+. The states also keep track of the c-command relations between the heads hosting the relevant licensor features. When a mover with f_i^- is encountered, the STA checks if the mover can satisfy any of the licensing requirements in component i. If so, those are removed from component i and all other components whose heads are c-commanded by the head for component i. At the end, no state may contain any non-empty components. This strategy reimplements licensing as a mechanism where the STA accrues "licensing debt" while moving through the tree. This debt has to be paid off by movers at a later point. The strategy works because the SpIC allows us to correctly synchronize the licensing debt across the states of all daughter nodes.

While each strategy is relatively simple on its own, integrating them is more difficult. An element may move to a higher position p from where it is only licensed by another element that moves to an even higher position. This requires keeping track of potential licensing debts for p which are then narrowed down to the actual licensing debt once it is known what actually moves to p, and then this debt must be paid off by whatever moves to a position above p. Further complicating the picture, some licensing requirements only need to be satisfied once (e.g. Principle A in the example above), whereas others hold throughout the derivation and are enforced after each movement step. The individual components of the automaton states thus must be synchronized in just the

right way, which complicates the construction of the STA even more.

6.2 Subcommand

It has been argued that some cases of long-distance binding involve *subcommand* instead of c-command (see Tang 1989 and Huang and Liu 2001, a.o.). A node x subcommands y iff x c-commands y or x is a specifier of some z that c-commands y. From the perspective of c-strings, x c-commands y iff there is some z in the c-string of y such that $x = z$ or x is the left sibling of a daughter of z. For instance, if Principle A in English allowed for subcommand instead of just c-command, then *John's picture pleases himself* would be well-formed as *John* is a specifier of the subject DP and thus subcommands *himself*.

Subcommand is beyond the reach of STAs because it makes the status of a node n dependent on the daughters of n's left siblings. Similar to the case of movement interactions, there are two possible replies to this. One could point out the rarity of subcommand, and that in the few cases where it arises, it serves as a means to furnish additional antecedents for reflexives beyond those that are already provided via c-command. Hence subcommand might be limited to the syntax-semantics interface and may not directly factor into licensing. Alternatively, one could simply increase the look-ahead of STAs from 1 to 2 so that the daughters of daughters are also taken into account. Subcommand then is just a more demanding case of c-command.

6.3 Extension to MG Derivation Trees

Once one equips STAs with a more powerful look-ahead mechanism to handle subcommand, MG derivation trees once again become a viable alternative to MG dependency trees. All the results in this paper extend from dependency tree to derivation trees if one generalizes STAs to deterministic top-down tree automata with finite look-ahead. That is not surprising because dependency trees can be converted to derivation trees by a linear tree transduction. This shows that derivation trees are the result of separating mothers and daughters in a dependency tree by a finitely bounded amount of material (Merge and Move nodes). Finite look-ahead is a means to reconstruct the relations of the dependency tree that have been obfuscated by this additional material.

It remains to be seen which one of the two representation formats ultimately provides for more insightful characerizations. One issue that derivation trees might shed some light on is the monotonic nature of c-command. With an STA over dependency trees, the daughters could be evaluated in any order to determine the states for a c-string constraint. While our current model proceeds left-to-right, we could have just as well gone right-to-left, inside out, or switched between those options based on the label of the mother. Yet only the first option corresponds to c-command as we know it. In a derivation tree, all information is conveyed via dominance. A specifier is not a sibling of the complement but rather resides in a structurally higher position (cf. the positions of *John* and the CP-complement in Fig. 2). Among all the command variants we just described, the empirically attested one in the form of c-command is the only one that is monotonic with respect to dominance.

6.4 Adjuncts

Another problem of dependency trees is the status of adjuncts. In order to obtain the correct c-command relations, an adjunct of node n would have to be treated as a left sibling of all the specifiers of n. But since adjunction is unbounded, this would mean that a node can have arbitrarily many daughters, whereas STAs are usually defined for trees with a finitely bounded arity. Adjuncts in derivation trees, on the other hand, do not create such issues because each adjunct grows the tree vertically rather than horizontally.

Either way, adjuncts must be subject to the same restriction as specifiers to preserve recognizability by STAs or a suitably generalized variant: even though an adjunct may move on its own, nothing may move out of an adjunct. This is of course a well-established property of adjuncts, known as the Adjunct Island Constraint. Once again, then, our specific subregular perspective derives a well-known limitation of movement.

The Adjunct Island Constraint has some principled exceptions such as *[which car]$_i$ did John drive Mary crazy while trying to fix* t_i (Truswell, 2007). It remains to be seen how these exceptions can be reconciled with our approach.

6.5 Across-the-Board Movement

Another well-known island constraint is the Coordinate Structure Constraint (CSC), which forbids extraction from a conjunct. The CSC itself is easy

enough to enforce with an STA. It is the exception to the CSC that is of interest here: extraction from a conjunct is permitted if extraction takes place from all conjuncts in the same coordination. Hence *which beer did Ed buy and Greg drink* is well-formed even though *which beer did Ed buy wine and Greg drink* is illicit. This is known as across-the-board movement (ATB).

Curiously, the ATB-exception is very natural from the STA perspective. STAs fail if one cannot clearly tell from the local configuration which one of several subtrees a move feature should be passed into. The SpIC and the Adjunct Island Constraint address this by excluding specifiers and adjuncts from this equation, leaving the complement as the only subtree that might contain additional movers. ATB-movement constitutes the opposite solution where the feature is instead passed into every subtree. This, too, is a fully deterministic process and thus within the limits of STAs. If even one conjunct did not need to contain a mover, then STAs would be faced with a non-deterministic choice that they cannot handle. While this has to be explored in greater detail based on a proper formalization of ATB-movement (e.g. Torr and Stabler 2016), it is remarkable that the abilities of STAs closely line up with the attested movement configurations.

6.6 Connection to Parsing

The preceding discussion shows that STAs not only furnish a tighter upper bound on syntactic complexity, they also explain core aspects of syntax: the importance of island constraints, the availability of ATB-movement, and the central role of c-command, which merely co-opts mechanisms that are independently needed for movement. But this in turn raises the fundamental question why STA-recognizability should be a relevant concept in syntax. We conjecture that STAs exhibit two properties that are attractive for parsing: top-down recognition, and determinism.

The highly predictive nature of human sentence processing suggests that some top-down strategy is employed, either directly in the form of recursive descent parser, or as a top-down filter of a left-corner parser. Given a choice between bottom-up or top-down recognition, the latter is a much more natural match for such parsing algorithms. Precompiling a top-down filter into the parse schema of an MG parser like the one in Stabler (2011b) or Stanojević and Stabler (2018) is not trivial, but feasible. Determinism ensures that this precompilation does not explode the parse space, which would slow down the parser. Hence STAs are a good match for current MG models of human sentence processing (Kobele et al., 2013; Gerth, 2015; Graf et al., 2017).

Of course a deterministic top-down automaton would also exhibit these properties, but these automata cannot even handle Merge, let alone Move. STAs are a minimal extension of deterministic top-down automata while also furnishing plenty of power for syntactic dependencies. They require some restrictions on movement, but each one of them also improves parsing performance by exponentially reducing the search space (see the discussion of the SpIC's impact on parsing performance in Stabler 2013).

If one assumes that the grammar is but a high-level description of the parser, the restriction to STAs may be an abstract counterpart to various parser constraints that are meant to improve efficiency. Subregular complexity in syntax may thus be closely connected to parsing.

7 Conclusion

We have shown that all current results on the subregular complexity of syntax are insightfully subsumed by sensing tree automata operating over MG dependency trees. The limits of STA-recognizability line up closely with well-established restrictions on movement and syntactic licensing conditions. A lot of issues remain to be formally worked out, but we are confident that the perspective developed in this paper will greatly expand our understanding of subregular syntax.

Acknowledgments

The work reported in this paper was supported by the National Science Foundation under Grant No. BCS-1845344. We thank the participants of the 2019 workshop on subregular complexity at Stony Brook University for their feedback. We are also grateful to the three anonymous reviewers, whose suggestions allowed us to streamline a lot of the formal definitions. In particular Reviewer 3 went above and beyond the call of duty.

References

Rolf Backofen, James Rogers, and K. Vijay-Shanker. 1995. A first-order axiomatization of the theory of finite trees. *Journal of Logic, Language and Information*, 4:5–39.

Hyunah Baek. 2018. Computational representation of unbounded stress: Tiers with structural features. In *Proceedings of CLS 53*, pages 13–24.

Chris Barker and Geoffrey K. Pullum. 1990. A theory of command relations. *Linguistics and Philosophy*, 13:1–34.

Patrick Blackburn, Claire Gardent, and Wilfried Meyer-Viol. 1993. Talking about trees. In *Proceedings of the Sixth Conference of the European Chapter of the Association for Computational Linguistics*, pages 30–36.

Marisa Ferrara Boston, John T. Hale, and Marco Kuhlmann. 2010. Dependency structures derived from Minimalist grammars. In Christian Ebert, Gerhard Jäger, and Jens Michaelis, editors, *MOL 10/11*, volume 6149 of *Lecture Notes in Computer Science*, pages 1–12. Springer, Berlin.

Noam Chomsky. 1995. *The Minimalist Program*. MIT Press, Cambridge, MA.

H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. TommasiK. 2008. Tree automata: Techniques and applications. Published online: `http://www.grappa.univ-lille3.fr/tata`. Release from November 18, 2008.

Thomas Cornell and James Rogers. 1998. Model theoretic syntax. In *The Glot International State of the Article Book*, volume 1 of *Studies in Generative Grammar 48*, pages 101–125. Mouton de Gruyter.

Robert Frank and K Vijay-Shanker. 2001. Primitive c-command. *Syntax*, 4(3):164–204.

Sabrina Gerth. 2015. *Memory Limitations in Sentence Comprehension. A Structure-Based Complexity Metric of Processing Difficulty*. Ph.D. thesis, University of Potdsam.

Saul Gorn. 1967. Explicit definitions and linguistic dominoes. In *Systems and Computer Science, Proceedings of the Conference held at University of Western Ontario, 1965*, Toronto. University of Toronto Press.

Thomas Graf. 2012a. Locality and the complexity of Minimalist derivation tree languages. In *Formal Grammar 2010/2011*, volume 7395 of *Lecture Notes in Computer Science*, pages 208–227, Heidelberg. Springer.

Thomas Graf. 2012b. Movement-generalized Minimalist grammars. In *LACL 2012*, volume 7351 of *Lecture Notes in Computer Science*, pages 58–73.

Thomas Graf. 2017. The power of locality domains in phonology. *Phonology*, 34:385–405.

Thomas Graf. 2018a. Locality domains and phonological c-command over strings. In *NELS 48: Proceedings of the Forty-Eighth Annual Meeting of the North East Linguistic Society*, volume 1, pages 257–270, Amherst, MA. GLSA.

Thomas Graf. 2018b. Why movement comes for free once you have adjunction. In *Proceedings of CLS 53*, pages 117–136.

Thomas Graf and Connor Mayer. 2018. Sanskrit n-retroflexion is input-output tier-based strictly local. In *Proceedings of SIGMORPHON 2018*, pages 151–160.

Thomas Graf, James Monette, and Chong Zhang. 2017. Relative clauses as a benchmark for Minimalist parsing. *Journal of Language Modelling*, 5:57–106.

Thomas Graf and Nazila Shafiei. 2019. C-command dependencies as TSL string constraints. In *Proceedings of the Society for Computation in Linguistics (SCiL) 2019*, pages 205–215.

Jeffrey Heinz, Anna Kasprzik, and Timo Kötzing. 2012. Learning in the limit with lattice-structured hypothesis spaces. *Theoretical Computer Science*, 457:111–127.

Jeffrey Heinz, Chetan Rawal, and Herbert G. Tanner. 2011. Tier-based strictly local constraints in phonology. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, pages 58–64.

Cheng-Teh James Huang and Cheng-Sheng Luther Liu. 2001. Logophoricity, attitudes and *ziji* at the interface. In Peter Cole, Gabrielle Herman, and Cheng-Tea James Huang, editors, *Long Distance Reflexives*, volume 33 of *Syntax and Semantics*, pages 141–195. Academic Press, New York.

M. A. C. Huybregts. 1984. The weak adequacy of context-free phrase structure grammar. In Ger J. de Haan, Mieke Trommelen, and Wim Zonneveld, editors, *Van Periferie naar Kern*, pages 81–99. Foris, Dordrecht.

Adam Jardine and Kevin McMullin. 2017. Efficient learning of tier-based strictly k-local languages. In *Proceedings of Language and Automata Theory and Applications*, Lecture Notes in Computer Science, pages 64–76, Berlin. Springer.

C. Douglas Johnson. 1972. *Formal Aspects of Phonological Description*. Mouton, The Hague.

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

Gregory M. Kobele. 2006. *Generating Copies: An Investigation into Structural Identity in Language and Grammar*. Ph.D. thesis, UCLA.

Gregory M. Kobele, Sabrina Gerth, and John T. Hale. 2013. Memory resource allocation in top-down Minimalist parsing. In *Formal Grammar: 17th and 18th International Conferences, FG 2012, Opole, Poland, August 2012, Revised Selected Papers, FG 2013, Düsseldorf, Germany, August 2013*, pages 32–51, Berlin, Heidelberg. Springer.

Gregory M. Kobele and Jens Michaelis. 2011. Disentangling notions of specifier impenetrability. In *The Mathematics of Language*, volume 6878 of *Lecture Notes in Artificial Intelligence*, pages 126–142, Berlin, Heidelberg. Springer.

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

Wim Martens. 2006. *Static Analysis of XML Transformation- and Schema Languages*. Ph.D. thesis, Hasselt University.

Wim Martens, Frank Neven, and Thomas Schwentick. 2008. Deterministic top-down tree automata: Past, present, and future. In *Proceedings of Logic and Automata 2008*, pages 505–530.

Kevin McMullin. 2016. *Tier-Based Locality in Long-Distance Phonotactics: Learnability and Typology*. Ph.D. thesis, University of British Columbia.

Jens Michaelis. 2004. Observations on strict derivational minimalism. *Electronic Notes in Theoretical Computer Science*, 53:192–209.

Jens Michaelis. 2009. An additional observation on strict derivational minimalism. In *FG-MOL 2005*, pages 101–111.

Jens Michaelis and Marcus Kracht. 1997. Semilinearity as a syntactic invariant. In *Logical Aspects of Computational Linguistics*, volume 1328 of *Lecture Notes in Artifical Intelligence*, pages 329–345. Springer.

Uwe Mönnich. 2006. Grammar morphisms. Ms. University of Tübingen.

Frank Morawietz. 2003. *Two-Step Approaches to Natural Language Formalisms*. Walter de Gruyter, Berlin.

James Rogers. 1998. *A Descriptive Approach to Language-Theoretic Complexity*. CSLI, Stanford.

James Rogers. 2003. Syntactic structures as multi-dimensional trees. *Research on Language and Computation*, 1(1):265–305.

James Rogers, Jeffrey Heinz, Gil Bailey, Matt Edlefsen, Molly Vischer, David Wellcome, and Sean Wibel. 2010. On languages piecewise testable in the strict sense. In Christan Ebert, Gerhard Jäger, and Jens Michaelis, editors, *The Mathematics of Language*, volume 6149 of *Lecture Notes in Artificial Intelligence*, pages 255–265. Springer, Heidelberg.

Sylvain Salvati. 2011. Minimalist grammars in the light of logic. In Sylvain Pogodalla, Myriam Quatrini, and Christian Retoré, editors, *Logic and Grammar — Essays Dedicated to Alain Lecomte on the Occasion of His 60th Birthday*, number 6700 in Lecture Notes in Computer Science, pages 81–117. Springer, Berlin.

Nazila Shafiei and Thomas Graf. 2019. The subregular complexity of syntactic islands. Ms., Stony Brook University.

Stuart M. Shieber. 1985. Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, 8(3):333–345.

Edward P. Stabler. 1997. Derivational Minimalism. In Christian Retoré, editor, *Logical Aspects of Computational Linguistics*, volume 1328 of *Lecture Notes in Computer Science*, pages 68–95. Springer, Berlin.

Edward P. Stabler. 2011a. Computational perspectives on Minimalism. In Cedric Boeckx, editor, *Oxford Handbook of Linguistic Minimalism*, pages 617–643. Oxford University Press, Oxford.

Edward P. Stabler. 2011b. Top-down recognizers for MCFGs and MGs. In *Proceedings of the 2nd Workshop on Cognitive Modeling and Computational Linguistics*, pages 39–48.

Edward P. Stabler. 2013. Two models of minimalist, incremental syntactic analysis. *Topics in Cognitive Science*, 5:611–633.

Miloš Stanojević and Edward Stabler. 2018. A sound and complete left-corner parser for Minimalist grammars. In *Proceedings of the 8th Workshop on Cognitive Aspects of Computational Language Learning and Processing*, pages 65–74.

Chih-Chen Jane Tang. 1989. Chinese reflexives. *Natural Language and Linguistic Theory*, 7:93–121.

John Torr and Edward P. Stabler. 2016. Coordination in Minimalist grammars: Excorporation and across the board (head) movement. In *Proceedings of the 12th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+12)*, pages 1–17, Düsseldorf, Germany.

Robert Truswell. 2007. Extraction from adjuncts and the structure of events. *Lingua*, 117:1355–1377.

Mai Ha Vu. 2018. Towards a formal description of NPI-licensing patterns. In *Proceedings of the Society for Computation in Linguistics (SCiL) 2018*, volume 1, pages 154–163. Article 17.

Mai Ha Vu, Nazila Shafiei, and Thomas Graf. 2019. Case assignment in TSL syntax: A case study. In *Proceedings of the Society for Computation in Linguistics (SCiL) 2019*, pages 267–276.