Deconstructing syntactic generalizations with minimalist grammars*

Marina Ermolaeva University of Chicago / Chicago, IL mail@mermolaeva.com

Abstract

Within the currently dominant Minimalist framework for syntax (Chomsky, 1995, 2000), it is not uncommon to encounter multiple proposals for the same natural language pattern in the literature. We investigate the possibility of evaluating and comparing analyses of syntax phenomena, implemented as minimalist grammars (Stabler, 1997), from a quantitative point of view. This paper introduces a principled way of making linguistic generalizations by detecting and eliminating syntactic and phonological redundancies in the data. As proof of concept, we first provide a small step-by-step example transforming a naive grammar over unsegmented words into a linguistically motivated grammar over morphemes, and then discuss a description of the English auxiliary system, passives, and raising verbs produced by a prototype implementation of a procedure for automated grammar optimization.

1 Introduction

Marr's (Marr, 1982) approach to understanding complex cognitive systems, including natural language, distinguishes between three levels of description: computational (abstract specification of what the system computes), *algorithmic* (structures representing the data and algorithms that manipulate them), and implementational (concrete realization of the algorithms in the hardware or wetware). Linguistic theories are considered to be at the computational level (Johnson, 2017) or between the computational and algorithmic level (Peacocke, 1986). In particular, syntactic literature often omits lower-level details such as full descriptions of units and operations involved. This allows for a bird'seye view of the problem but leads to a lack of explicitness. While the body of work in this field holds tremendous empirical and theoretical value, there is no straightforward way to extract from it a

*This paper is based on a chapter of the author's dissertation (Ermolaeva, 2021). consensus on what constitutes the best description (grammar) for any given set of language data.

How can one choose between various structures that could underlie the same linguistic construction based on some robust quantitative metric and in a way compatible with the insights of theoretical syntax? First, this task requires translating the data into an explicit formal representation. For this purpose, minimalist grammars (Stabler, 1997) are a natural choice. Designed as a formalization of the Minimalist Program (Chomsky, 1995, 2000), they are compatible with analyses of syntax phenomena found in theoretical literature and provide a way to implement them as grammar fragments, explicitly defining syntactic units and operations.

There exists a substantial amount of work on learning formal grammars from unstructured strings; see e.g. (Clark and Eyraud, 2007; Yoshinaka, 2011; Clark, 2017). These techniques are based on the notion of distributional similarity: in short, words found in the same contexts are assigned the same syntactic category. Resulting algorithms can make plausible generalizations based on observable data; however, they are of limited applicability if the goal is to produce results similar to those of theoretical syntax, which is heavily reliant on highly abstract concepts. Two broad considerations are particularly illustrative in this respect:

• Empty categories. Null DPs of various flavors, null heads such as complementizers, traces/unpronounced copies are commonplace in the literature, even though none of them are directly visible in the raw data. However, works on grammar induction tend to avoid silent elements altogether. For example, classical categorial grammars in (Kanazawa, 1995) are defined over an alphabet of constant symbols and cannot derive the empty string, and (Clark and Eyraud, 2007; Yoshinaka, 2011) explicitly prohibit rules with empty right-hand sides. • Morphology. Many crucial generalizations require words to have internal structure. For example, the standard analysis of passive constructions relies on the object being promoted into the subject position. In order for this to work, the verb has to consist of at least two elements: the root and the passive morpheme. Any learning algorithm incapable of manipulating units smaller than a word will miss this generalization.

In order to focus on these and similar issues, rather than use unstructured sentences as input, we start with a grammar defined over unsegmented words and in a maximally theory-neutral way. The goal is to refine it, producing an intuitively correct description accounting for the original data. Unlike grammar induction, the focus is on capturing generalizations within the grammar rather than generalizing beyond the corpus; in fact, the output grammar is expected to generate the same string language as the input. To highlight this distinction, we refer to this task as *grammar optimization*.

2 Minimalist grammars

We use a version of minimalist grammars (MGs) primarily based on (Stabler, 1997, 2001). In order to define an MG, one has to fix an alphabet of phonological segments Σ and a set of feature names *Base* and to define a lexicon *Lex* – a finite set of *lexical items* (LIs). An LI is a pair consisting of a phonological exponent and a sequence of syntactic features.¹ Each syntactic feature is characterized by its name (drawn from *Base*) and type, as shown in (1).

		Attractor	Attractee
(1)	merge	=x (right selector)	x (category)
		x= (left selector)	
		=>x (HM selector)	
	move	+x (overt licensor)	-x (licensee)
		*x (covert licensor)	

Selectors and licensors together are called *attractors*, and categories and licensees are *attractees*. The first feature of each LI is accessible to the structure building operations, **merge** and **move**, that consume features with matching names to generate complex expressions from *Lex*:

- merge is a binary operation that targets selectors and categories, combining two syntactic expressions into a new one. The dependent is merged on the left if the selector is of the form x=, and on the right if it is of the form =x, where x ∈ Base;
- merge with head movement (HM) is triggered by selectors of the form =>x. It proceeds as right merge and concatenates the heads of its arguments, treating the head carrying the =>x as a suffix. This operation offers a way of forming complex morphological words;
- move matches a licensor with a licensee within the same expression. Overt licensors (+x) cause the moving sub-expression to become a (left) sister of the head, leaving behind an empty node ε without a string component or syntactic features. Covert move (*x) leaves the string component behind.

The expression with the attractor becomes the *head* of the new expression; and the one with the attractee becomes its *dependent*. We label the parent node with < if the head is on the left or > if the head is on the right. Matched features (indicated as x) are deleted, making the next feature in the sequence accessible for syntactic operations. The dependent introduced by the first attractor of an LI is its *complement*, and all subsequent dependents are *specifiers*.

A *complete expression* of category \times has no unchecked features left, except for \times on its head. We will be primarily concerned with complete expressions of category t (for Tense) and their string yields (*sentences*). Note that, in order to arrive at a complete expression via **merge** and **move**, all LIs involved must have feature sequences where all licensors and selectors precede the (unique) category feature, and all licensees follow it.

Consider two toy grammars (2), which generate the same set of sentences:

	a.	Mary :: d-k	b. <i>Mary</i> :: d -k
		<i>laughs</i> :: =d+kt	<i>-s</i> ∷ =>v+kt
(2)		<i>laughed</i> :: =d+kt	<i>-ed</i> :: =>v+kt
		<i>jumps</i> :: =d+kt	<i>laugh</i> ∷ =d v
		<i>jumped</i> :: =d+kt	<i>jump</i> :: =d v

In linguistic terms, (2a) treats each sentence as a single $\pm P$ headed by an unsegmented verb (3a). In (2b) finite verbs are complex heads formed by head movement. The lexical verb directly selects

¹While this formalism ignores semantic features, there is work on adding semantics to MGs, e.g. (Kobele, 2012). Augmenting the proposed approach with semantic information is a potential direction of future work.

its argument and forms a vP, while the suffix takes the vP as its complement and is responsible for moving the subject into its specifier position (3b).

(3) Structure of *Mary laughs* generated by (2a) and (2b)



A convenient way to visualize head-complement relations within a lexicon is a directed multigraph where vertices are category features, and each edge corresponds to an LI and connects the category of its complement to its own category. It ignores specifiers formed by **merge** and **move**, focusing on the subset of relations which are relevant for morphologically complex words.

(4) (2a) and (2b) as multigraphs



Intuitively, (2b) is an improvement over (2a). By recognizing internal structure within verbs, it captures similarities found within verbal paradigms (*laughs, laughed* vs. *jumps, jumped*) and across paradigms (*laughs, jumps* vs. *laughed, jumped*). On the other hand, (2a) misses all these generalizations. To account for a new verbal paradigm (e.g. *smiles, smiled*), we would need to add two new LIs to (2a), but only one to (2b).

How to express this intuition in quantitative terms? One common approach is based on the Minimum Description Length principle (MDL, Rissanen 1978). Within linguistics, MDL has been used,

among other examples, for induction of phonological constraints (Rasin and Katzir, 2016) and ordered rules (Rasin et al., 2018), morphological segmentation (Goldsmith, 2001, 2006), and inferring syntactic categories given known morphological patterns (Hu et al., 2005). In this framework, the best grammar to describe a data corpus is the one that minimizes the sum of the following, both measured in bits: (i) size of the grammar and (ii) size of the corpus, as encoded by the grammar. An overfitting grammar, which makes few (or no) generalizations on the data, would have a very large grammar size and small (or zero) corpus size. At the other end of the spectrum, an overgenerating grammar, capable of producing any string at all, would itself be very small but require a lengthy encoding of the corpus. MDL strikes a balance by taking both considerations into account. We focus on grammar size here and will return to corpus size and the problem of overgeneration in section 5.

To encode a minimalist lexicon, we use a straightforward encoding scheme (adapted from Katzir 2014) which treats each LI as a sequence of symbols followed by a separator. Each syntactic feature is considered two symbols (corresponding to its type and name). Letting T denote the set of syntactic feature types, the size of a lexicon *Lex* over a set of categories *Base* is given by

$$\underbrace{\sum_{\substack{s::\delta \in Lex \\ \text{total number of symbols}}} (|s|+2 \times |\delta|+1) \times \underbrace{\log_2(|\Sigma|+|T|+|Base|+1)}_{\text{cost of encoding per symbol}}.$$

We fix Σ and T, with |T| = 7 and $|\Sigma| = 26$ (Σ being the set of English letters). Then the above is a function of the number of LIs and the following three metrics:

- |*Base*|, the number of unique category features in *Lex*;
- $\sum_{\text{syn}} = \sum_{s::\delta \in Lex} (|\delta|)$, the total count of syntactic features in *Lex*;
- $\sum_{\text{phon}} = \sum_{s::\delta \in Lex} (|s|)$, the total length of all string components in *Lex*.

Table (5) presents the values of these metrics for (2a) and (2b):

		Base	\sum_{syn}	\sum_{phon}	Grammar (bits)
(5)	(2a)	3	14	28	317.78
	(2b)	4	12	16	236.16

Separating roots and suffixes in (2b) comes at the cost of an extra category feature. However, this pays off by eliminating redundant strings, which reduces \sum_{phon} . Moreover, four instances of +k are collapsed into two, yielding a small reduction of \sum_{syn} . These differences would be even more noticeable with a larger dataset, especially with respect to open-class words, since adding a new verb to (2a) would have a higher cost (in both syntactic features and string components) compared to (2b).

3 Decomposing lexical items

How can we transition from a grammar over words such as (2a) to a grammar over morphemes such as (2b)? One way is to directly modify LIs while keeping track of how the changes affect all structures that use them. For example, we can split the string component of *laughed* :: =d +k t (6a) into two substrings, *laugh* and *ed*, and its feature sequence into =d and +k t. We then assemble two useful LIs from these elements by introducing x, a category feature that is *fresh* (i.e. not used before in the grammar), with head movement to combine them into a morphological word (6b).

(6)



The same operation can be applied even if some or all of the splits result in empty sequences. For instance, we can split *laugh* :: $=d \times by$ assigning its entire string component to one LI, and its feature sequence to another (6c). This creates a root with only the new feature y and shifts its selectional properties to an empty head. In each case, the new LIs generate the same strings as the original ones.

Following (Kobele, 2018, to appear), we define the operation of *lexical item decomposition* as a generalization of this idea. Consider an arbitrary LI $w :: \alpha\beta x\gamma$, where $w \in \Sigma^*$, α and β are sequences of licensors and selectors, $x \in Base$, and γ is a sequence of licensees. Then it can be decomposed as follows:

$$u ::: \alpha y$$
$$w ::: \alpha \beta x \gamma \qquad \cdots \rightarrow \qquad -v ::= > y \beta x \gamma$$
$$w = u \oplus v,$$

where y is a fresh feature name added to *Base*. Since the resulting structure has to behave as a single unit with respect to **move**, the split must take place to the left of the category feature. The original LI is replaced with two new items, whose string components are *u* and *v*, along with a *morphological rule* generating *w* from *u* and *v* via some operation \oplus .

Lexical item decomposition formalizes discovering structure within words. However, it cannot transform a grammar like (2a) into one like (2b) on its own. First, every split creates a fresh feature name, so each newly formed affix is only compatible with a single stem. Second, decomposing LIs in isolation is very permissive, and whether a given step is reasonable depends on other items in the lexicon. To recognize useful steps, decomposition needs to process multiple LIs simultaneously, using similarities between them to inform its decisions. We will refer to this strategy as *batch decomposition*.

In the rest of this section we build and expand on Kobele's work to define a toolkit of operations over MGs. We adopt a simple model of morphology that defines \oplus as string concatenation, which renders morphological rules trivial. It is sufficient for cases like (6b), where *w* is cleanly separated into a stem and a suffix, or (6c), where the string component is not split at all, but not for irregular forms such as *was* or *ran*.

Consider the transition in (7). It starts with a batch of four LIs and factors out the elements they have in common: the prefix *laugh*- and the syntac-

tic feature =d. These repeated elements are then expressed as a single new LI, $laugh :: =d \times$, adding only one new feature to *Base*.

(7)
$$laugh :: = dv$$

$$laughed :: = d+kt$$

$$laughing :: = dprog$$

$$laughs :: = d+kt$$

$$\begin{cases} \\ -\epsilon :: =>xv \\ -ed :: =>x+kt \\ -ing :: =>x+kt \\ -s :: =>x+kt \end{cases}$$

This is an example of **left decomposition**, as the shared part of both the string component and feature sequence is on the left-hand side. Similarly, **right decomposition** factors out right-hand side commonalities. In a (primarily) suffixal language such as English, left decomposition can be used to identify stems, and right decomposition to identify affixes.

LIs of the form $-\epsilon :: =>x \lor for any x, \lor \in Base$, which we will refer to as category changers, are of special interest. In the general case, $-\epsilon ::= x y$ represents the idea that expressions of category v have a more limited distribution than those of category x; that is, any x can freely become an y, but not necessarily the other way around. That said, some category changers can be eliminated while still allowing the grammar to generate all sentences it did before. For example, the distinction between x and v in (7) can be safely collapsed, essentially reversing the effects of decomposition for *laugh* :: =dv. Another redundant configuration involves alternative paths through the grammar multigraph consisting exclusively of category changers. In order to deal with these, we introduce two auxiliary operations that target and delete category changers and are convenient to describe in (multi)graph terms: contraction, which removes an edge and merges the two vertices it previously joined, and **deletion**, which removes an edge as long as there is another path of category changers from its origin to its destination vertex.

4 Towards a grammar over morphemes

The following example shows how the operations introduced in the previous section can be used to transform a naive word-based grammar into one over morphemes by making generalizations about word structure. We start with a small lexicon of unsegmented words encoding a fragment of the English auxiliary system (8).

(8) Original lexicon



Focusing on syntax and morphotactics, we abstract away from morphological irregularities of English and replace each word with a sequence of morphemes formed by string concatenation; in particular, *is* is rendered as *bes*, and *will* as *wills*. We continue using |Base|, \sum_{syn} , and \sum_{phon} , along with grammar size in bits, to keep track of how each transition changes the complexity of the lexicon.

We begin by decomposing lexical verbs. The three LIs targeted in (9) share the prefix *laugh*- and the first syntactic feature =d. Batch decomposition factors out these commonalities, associating them with the fresh feature f1 (highlighted in boldface).

(9) left decomposition of laugh :: =d+kt, laughing :: =d q, laugh :: =d v



Similarly, (10) decomposes the forms of *jump*, introducing another fresh feature, £2. The lexicon now contains two copies each of *-s*, *-* ϵ , and *-ing*.

(10) left decomposition of jumps :: =d+kt, jumping :: =d q, jump :: =d v

•		-
Mary	:: d-k	-€ ∷ =>f1v
bes	:: =g+kt	<i>laugh</i> ∷ =d f1
wills	:: =v+kt	-s :: => f2 +kt
be	:: =g v	-ing :: => f2 g
- <i>S</i>	:: =>fl+kt	-€ :: => f2 ∨
-ing	:: =>flg	<i>jump</i> :: =d f2



The two instances of *-ing* share the entire string component, and we can associate their syntactic differences, $=> \pm 1$ vs. $=> \pm 2$, with empty heads (11). Each decomposition step made so far immediately pays off in terms of \sum_{phon} , but adds a new category to *Base* and increases \sum_{syn} .

(11) right decomposition of -ing :: => flg,

```
-ing :: =>f2 g
```



Step (12) is an instance of **contraction**, which collapses remaining distinctions between *laugh* and

jump. Even though *jump* can now be selected by suffixes previously compatible only with *laugh*, and vice versa, no new morphological words are created. The new projection introduced by f4 now hosts both lexical verbs.

(12) contraction of $-\epsilon :: => f1 f3, -\epsilon :: => f1 f2$



Note that **contraction** in general has a potential for overgeneration. For example, contracting $-\epsilon ::$ =>f1 v in (11) instead would have created a path from v back to f3, generating the infinite set of ungrammatical sentences **Mary wills be* (*being*)⁺ *laughing*.

The LIs targeted in (13) share the suffix -s and the sequence of syntactic features +k t. By factoring out these commonalities, we create what is essentially a dedicated Tense projection. Since the batch shares more than one syntactic feature, this decomposition step immediately reduces \sum_{syn} and keeps \sum_{phon} the same.



This grammar still contains two partially redundant instances of *be*. We decompose them in (14).





The following **contraction** step (15) brings |Base| back to 7 and \sum_{syn} back to 21, same as in (13), before decomposing *be*.

(15) contraction of $-\epsilon :: = f \in v$

Mary :: d-k	-€ ∷ =>f4 f5
<i>-s</i> :: =>f5+kt	<i>-ing</i> :: =>f4g
-€ ∷ => f7 f5	-e ∷ =>f4 f7
<i>be</i> :: =g f7	<i>laugh</i> ∷ =d f4
<i>will</i> :: = f7 f5	<i>jump</i> :: =d f 4



Collapsing f6 and v into f7 in (15) has created a path of category changers from f4 to f5, which opens the path for **deletion** (16) of the empty LI directly connecting these two categories, - $\epsilon :: => f4 f5$. This last step finally pushes \sum_{syn} below the value it had in (13). The decision to decompose the two instances of *be* rather than leave them intact took two more steps to fully pay off.

(16) **deletion** of $-\epsilon :: => f4 f5$





Figure 1: Input (7572.80 bits)



Figure 2: Output (1964.68 bits)

The final lexicon (16) produces more complex structures, treating each verb as a multimorphemic word. It captures a number of correct generalizations: the roots in both verbal paradigms have been identified and separated from inflectional morphology, and they share the same syntactic category. A lexical verb can be selected (directly or via category changers) by *-ing*, *will*, or *-s*, whereas a phrase headed by *will* can only be selected by *-s*. These improvements are reflected in the metrics (17): the new lexicon beats the original in terms of both \sum_{syn} and \sum_{phon} , at the expense of two features added to *Base*.

		Base	\sum_{syn}	\sum_{phon}	Grammar
(17)	Original (8)	5	24	49	565.54
	Final (16)	7	19	23	375.03

5 Automated decomposition

In the previous section we has shown how a linguistically plausible grammar over morphemes can, in principle, be obtained from a naive one over unsegmented words – or, in other words, that the desired output is in the search space defined by the operations introduced in section 2. The next logical step is to automate the process of finding it.

To this end, we have developed a prototype Python implementation of an optimization procedure for MGs.² It starts with a grammar over words, similar to the one in (8), and uses beam search to navigate the space of grammars. While MDL is a natural choice for a cost function to evaluate candidate grammars, we have found that this metric as typically defined does not perform as expected on smaller grammars generating finite languages (due to its sensitivity to relative sizes of the grammar and corpus), and its variations that put additional emphasis on corpus cost lead to under-generalization and worse overall results.³ Because of this, our implementation uses grammar size in bits as the cost function. To control overgeneration, instead of corpus size, it utilizes a heuristic constraining the application of **contraction**.⁴

When applied to the lexicon in (8), the procedure does converge exactly to a grammar identical to (16) up to feature names. This is both expected and encouraging, as it indicates a connection between conciseness (as defined by grammar size) and naturalness of a grammar.

A larger sample input and output are given as multigraphs in Figures 1 and 2, respectively. The output compresses the auxiliary system, making use of category changers to implement hierarchical relations between categories. It collapses each verbal paradigm into a single lexical verb assigned a feature sequence according to its distribution: intransitive (dance, laugh...), transitive (praise, like...), verb with a clausal complement (declare, think...), or raising verb (seem, appear). All proper nouns and most lexical verbs carry a single category feature, and their syntactic requirements are supplied by LIs that select them. This is consistent with the idea of acategorial roots having to merge with a categorizing head, which is a general assumption in Distributed Morphology (Marantz, 1997; Embick and Marantz, 2008) and, from the

²The code and supplementary materials can be found at https://github.com/mermolaeva/mg-optimizer.

³See (Ermolaeva, 2021, pp.112–119) for additional details and discussion.

⁴Informally, this heuristic rules out candidate grammars generating morphological words (complex heads) that differ, either phonologically or syntactically, from those present in the original grammar.

quantitative perspective, a beneficial strategy especially for open-class items.

In general, we observe a tendency to push syntactic differences between LIs into empty functional heads. As a concrete illustration, consider the raising verb *seem*. The input includes a separate LI for each of its forms; moreover, *seem* in constructions with raising and expletive *it* is treated as unrelated LIs. For instance, the derivations of *Mary seems to smile* (3a) and *It seems that Mary smiles* (3b) involve, respectively, *seems* :: =to +kt and *seems* :: =c1 +kt.



Figure 3: Examples before optimization

After optimization (4) both paradigms are collapsed into a single root LI carrying just the category feature: *seem* :: 6. The distinction between the the verbs' selection requirements (=to vs. =c1) is now carried by empty categorizing heads which take the root as their complement: $-\epsilon :: =>6=to 3$ and $-\epsilon :: =>6=c1 3$. The suffixes of the original verbs, in turn, are factored out and unified with those of other verbal paradigms – for example, the output grammar contains a single present tense suffix -s :: =>1 +k t.



Figure 4: Examples after optimization (partial trees shown for space considerations)

6 Conclusion

This paper outlines an approach to formalizing and automating the intuition behind generalizations in natural language syntax. Building upon the idea of lexical item decomposition, we have defined a set of basic operations over minimalist lexica and demonstrated how complex, intuitively correct and quantitatively optimal analyses arise through repeated application of easily interpretable steps.

From a broader perspective, the intent of this project is twofold. On the one hand, it seeks to enable computational learning of natural language to make use of the insights and theoretical devices developed in the linguistic literature. The second goal is to show how mainstream syntax can benefit from a formally explicit approach. In this paper we have taken a step towards both by capturing some aspects of *intuitive goodness* of syntactic descriptions and connecting it to the more easily definable notion of *quantitative goodness*.

References

- Noam Chomsky. 1995. *The Minimalist Program*. MIT Press, Cambridge, MA.
- Noam Chomsky. 2000. Minimalist Inquiries: the framework. In Roger Martin, David Michaels, and Juan Uriagereka, editors, *Step by Step: Essays on Minimalist Syntax in Honor of Howard Lasnik*, pages 89–156. MIT Press, Cambridge, MA.
- Alexander Clark. 2017. Computational learning of syntax. Annual Review of Linguistics, 3:107–123.
- Alexander Clark and Rémi Eyraud. 2007. Polynomial identification in the limit of substitutable context-free languages. *Journal of Machine Learning Research*, 8(Aug):1725–1745.
- David Embick and Alec Marantz. 2008. Architecture and blocking. *Linguistic inquiry*, 39(1):1–53.
- Marina Ermolaeva. 2021. Learning syntax via decomposition. Ph.D. thesis, University of Chicago.
- John Goldsmith. 2001. Unsupervised learning of the morphology of a natural language. *Computational linguistics*, 27(2):153–198.
- John Goldsmith. 2006. An algorithm for the unsupervised learning of morphology. *Natural language engineering*, 12(4):353–372.
- Yu Hu, Irina Matveeva, John Goldsmith, and Colin Sprague. 2005. Using morphology and syntax together in unsupervised learning. In *Proceedings of the Workshop on Psychocomputational Models of Human Language Acquisition*, pages 20–27. Association for Computational Linguistics.
- Mark Johnson. 2017. Marr's levels and the minimalist program. *Psychonomic bulletin & review*, 24(1):171–174.
- Makoto Kanazawa. 1995. Learnable Classes of Categorial Grammars. Ph.D. thesis, Stanford University, Stanford, CA, USA. UMI Order No. GAX94-29947.
- Roni Katzir. 2014. A cognitively plausible model for grammar induction. *Journal of Language Modelling*, 2.
- Gregory M Kobele. 2012. Importing montagovian dynamics into minimalism. In International Conference on Logical Aspects of Computational Linguistics, pages 103–118. Springer.
- Gregory M. Kobele. 2018. Lexical decomposition. *Computational Syntax* lecture notes.
- Grerory M. Kobele. to appear. Minimalist grammars and decomposition. In Kleanthes K. Grohmann and Evelina Leivada, editors, *The Cambridge Handbook* of Minimalism. Cambridge University Press, Cambridge.

- Alec Marantz. 1997. No escape from syntax: Don't try morphological analysis in the privacy of your own lexicon. *University of Pennsylvania working papers in linguistics*, 4(2):14.
- David Marr. 1982. Vision: A Computational Investigation into the Human Representation and Processing of Visual Information. Henry Holt and Co., Inc., New York, NY, USA.
- Christopher Peacocke. 1986. Explanation in computational psychology: Language, perception and level 1.5. *Mind & language*, 1(2):101–123.
- Ezer Rasin, Iddo Berger, Nur Lan, and Roni Katzir. 2018. Learning phonological optionality and opacity from distributional evidence. In *Proceedings of NELS*, volume 48, pages 269–282.
- Ezer Rasin and Roni Katzir. 2016. On evaluation metrics in optimality theory. *Linguistic Inquiry*, 47(2):235–282.
- Jorma Rissanen. 1978. Modeling by shortest data description. *Automatica*, 14(5):465–471.
- Edward P. Stabler. 1997. Derivational minimalism. In Christian Retoré, editor, Logical Aspects of Computational Linguistics: First International Conference, LACL '96 Nancy, France, September 23–25, 1996 Selected Papers, pages 68–95. Springer Berlin Heidelberg, Berlin, Heidelberg.
- Edward P. Stabler. 2001. Recognizing head movement. In *Proceedings of the 4th International Conference on Logical Aspects of Computational Linguistics*, LACL '01, pages 245–260, London, UK, UK. Springer-Verlag.
- Ryo Yoshinaka. 2011. Efficient learning of multiple context-free languages with multidimensional substitutability from positive data. *Theoretical Computer Science*, 412(19):1821–1831.