The Influence of Context on the Learning of Metrical Stress Systems Using Finite-State Machines

Cesko Voeten
Leiden University
c.c.voeten@hum.leidenuniv.nl

Menno van Zaanen
Tilburg University
mvzaanen@uvt.nl

Languages vary in the way stress is assigned to syllables within words. This article investigates the learnability of stress systems in a wide range of languages. The stress systems can be described using finite-state automata with symbols indicating levels of stress (primary, secondary, or no stress). Finite-state automata have been the focus of research in the area of grammatical inference for some time now. It has been shown that finite-state machines are learnable from examples using state-merging. One such approach, which aims to learn k*-testable languages, has been applied to stress systems with some success. The family of* k*-testable languages has been shown to be efficiently learnable (in polynomial time). Here, we extend this approach to* k,l*-local languages by taking not only left context, but also right context, into account. We consider empirical results testing the performance of our learner using various amounts of context (corresponding to varying definitions of phonological locality). Our results show that our approach of learning stress patterns using state-merging is more reliant on left context than on right context. Additionally, some stress systems fail to be learned by our learner using either the left-context* k*-testable or the left-and-right-context* k,l*-local learning system. A more complex merging strategy, and hence grammar representation, is required for these stress systems.*

1. Introduction

The stress systems of the languages of the world show a huge amount of variation and diversity, but at the same time display remarkable unity with respect to certain properties (cf. Goedemans and van der Hulst 2013). Consider, for instance, example words

Submission received: 1 March 2016; revised version received: 19 July 2017; accepted for publication: 1 March 2018.

doi:10.1162/COLI_a_00317

including stress information from the following two languages: Icelandic (Example (1)) and Yurakaré (Example (2)):

(1) *Icelandic* [Árnason 1985]

1.	tás.ka	'briefcase'
2.	høf.ðiŋ.ja	'chieftain' (gen. pl.)
3.	á.kva.rɛ̀l.la	'aquarelle'
4.	bí.o.grà.fi.a	'biography'

(2) *Yurakaré* [van Gijn 2006]

1.	ku.dá.wa	'lake'
2.	ti.pó.jo.re	'my canoe'
3.	ma.là.ŋo.lé.ju	'don't fall in love with him'

These languages, although unrelated, display some similar grammatical properties. First, we note that in both languages, all words contain a single primary stress (marked by the uptick 'ˊ') and possibly one secondary stress (marked by the downtick 'ˋ'). (Longer words will contain more secondary stresses; other languages may also allow tertiary or quaternary stress marks.) Second, observe that two stress marks are always separated from each other by a single syllable—stress "clashes" are not permitted in either language. Third, note that in both languages, the final syllable in a word is never stressed, even if this results in a break of the stressed-unstressed cadence.

Of course, none of these generalizations (based on a sample of only two languages!) are categorical absolutes; there are many languages that do not abide by all of them, some even by any of them. Also, within languages, these generalizations do not hold absolutely; Yurakaré, for instance, allows for a prefix 'tá-' 'our', which is always stressed, which can break the canonical stressed-unstressed pattern. Even in the shadow of idiosyncrasies like these, we can say that there generally tends to be a great degree of structure to stress systems in languages; there are only a small handful of languages (e.g., Maskikit-Essed and Gussenhoven 2016) that truly do not have any kind of metrical stress structure.

The overall similarities between the many different stress systems of the languages of the world (we refer to Hayes [1995] for a very thorough overview of the many cross-linguistic similarities and differences) suggest that there is a certain set of properties favored by all languages. One of these properties, noted by Heinz (2007, 2009), is that all stress patterns are **neighborhood-distinct**. From an automata-theoretic point of view, neighborhood-distinctness means that for all states in an acceptor, the states that represent a specific combination of a stress type and its direct context can only occur in the acceptor once. From a phonological point of view, this observation corresponds closely to the phonological notion of locality, standardly formulated as "linguistic rules do not count beyond two" (Kenstowicz 1994, page 597). In the present context of stress

systems, this means that a rule parsing syllables into stressed feet should only be able to refer to any one of a syllable's two immediately adjacent neighbors. The observation by Heinz is that any such combination of three syllables should only occur once in the phonological grammar, if Kenstowicz was correct in defining locality based on a window of three syllables. Taken together, these observations offer a starting point for creating an artificial learner of metrical stress structure in terms of finite-state automata, which is precisely the effort that the present article undertakes. The key variable that is investigated is the size of the context over which the learner will generalize: How much context should the learner take into account before making a generalization about a grammatical pattern? Phonologically speaking, this question seeks an empirical definition of locality; in automata-theoretic terms, the question asks how much overlap there needs to be between two different substrings before a learner can induce that a token in both substrings corresponds to a common state in an acceptor.

2. Computational Background

The property that all stress systems appear to be neighborhood-distinct suggests that they could be efficiently learnable by an artificial learner. Consider a language like Pintupi (Hansen and Hansen 1969), which assigns stress on the basis of trochaic feet, starting on the left edge of the word, in a quantity-insensitive manner (the stress pattern does not make a distinction between different syllable weights):

(3) ká.tu 'above'

(4) tʲú.ta.ja 'many'

(5) pú.l̪iŋ.kà.la.tʲu 'we (sat) on the hill'

(6) tʲá.mu.lì̀m.pa.tʲùŋ.ku 'our relation'

(7) ʈ́i.l̪i.rì̀.ŋu.làm.pa.tʲu 'the fire for our benefit flared up'

(8) kú.ra.nʲù.lu.lì̀m.pa.tʲù.ɻa 'the first one (who is) our relation'

When converted into a purely syllable-based form, the stress types shown in Examples (3–8) can be represented as follows:

(9) (σ́σ)

(10) (σ́σ)σ

(11) (σ́σ)(σ̀σ)σ

(12) (σ́σ)(σ̀σ)(σ̀σ)

(13) (σ́σ)(σ̀σ)(σ̀σ)σ

(14) (σ́σ)(σ̀σ)(σ̀σ)(σ̀σ)

Note that the representations in Examples (9–14) can be captured by a single regular expression:

(15) $(\acute{\sigma}\sigma) + [(\grave{\sigma}\sigma)] * + [\sigma]$

where we denote the grouping operator using square brackets to not clash with the parenthetical notation that is used to indicate foot structure.

The observation by Heinz (2009), which we may repeat from the pattern formula in Example (15), is that each and every component of a regular expression describing a stress grammar should be unique. That is, if the neighborhood were defined as a single syllable, we would not predict to find a language that uses the same building blocks $\{(\acute{\sigma}\sigma), [(\grave{\sigma}\sigma)]_0, [\sigma]\}$, but in a pattern like Example (16), where the same block is used at multiple positions in the formula:

(16) $(\acute{\sigma}\sigma) + [(\grave{\sigma}\sigma)] * + [\sigma] + [(\grave{\sigma}\sigma)] *$

It is entirely possible for such a language to exist and be learned, but this must mean that such a learner is using a different (i.e., wider) syllable span for its definition of phonological neighborhood. A learner assuming a neighborhood to equate a single adjacent syllable would not be able to optimally represent the grammar. This is because in the representation offered in Example (16), a two-syllable substring 'σσ̀' (which would then constitute a complete neighborhood) can be ascribed to *two*, as opposed to one, substrings of the general pattern (indicated by the underlined symbols):

(17) $(\acute{\sigma}\underline{\sigma) + [(\grave{\sigma}}\sigma)] * + [\sigma] + [(\grave{\sigma}\sigma)] *$

(18) $(\acute{\sigma}\sigma) + [(\grave{\sigma}\sigma)] * + [\underline{\sigma] + [(\grave{\sigma}}\sigma)] *$

The observation by Heinz (2009) is that such (in a way, ambiguous) patterns are not attested in the world's languages. The reason, Heinz argues, is that the language learner is not able to successfully distinguish between the two possibilities (Examples (17) and (18)), and is therefore biased towards learning a grammar in which the problematic expression 'σσ̀' uniquely maps onto a *single* substring of the grammar. In other words, the learner is intrinsically biased toward neighborhood-distinctiveness, and for this reason languages that do not obey this requirement are not learnable, and hence, non-existent.

A major question is thus what the most suitable definition of a neighborhood should be. This can be operationalized as a question about two properties of phonological neighborhood: How large should it be, and in what direction should it be evaluated? The former issue is a question of how we should define the phonological notion of locality: If we are going to impose, on all (possible) languages, a phonological restriction that every substring of the overall stress pattern must be unique, how big should these substrings be? Under Kenstowicz's (1994) notion of locality, the answer would be three syllables: Every one of the substrings to consider consists of one syllable, plus an additional syllable to the left (defined by $k = 1$, for the size of the left context) and an additional syllable to the right ($l = 1$, describing the size of the right context). This provides a three-syllable window—a defining entity in analyses of at least West-Germanic stress systems (e.g., Kager 1989; van Oostendorp 2012). Languages that are

learnable by a finite-state-automaton learner using this notion of locality as its criterion for determining whether two states in a prefix tree are equivalent (and hence mergeable) are termed k, l**-local**, where k and l are coefficients determining how much context (in our case, how many syllables) to the left and the right, respectively, are considered at each edge of a symbol corresponding to a state.

Even though it may seem that learning a finite-state machine describing a k, l-local language might describe the same language as a finite-state machine that describes a k-testable language (with the latter language's k equal to the former language's $k + l$), it is important to observe that based on the same training data, different languages may be learned. Learning a finite-state machine that describes a two-testable language based on the following data:

(19) ὸὸ

(20) ὸσὸσό

results in the finite-state machine depicted in Figure 1, whereas a finite-state machine that learns a 1, 1-local language based on these data is found in Figure 2. Note that the finite-state machine in Figure 2 can generate 'ὸσὸὸ', which the finite-state machine in Figure 1 cannot.

It is important to note that k-testable languages are proved to be efficiently learnable (Garcia and Vidal 1990). However, this is not to say that a k-testable learning approach actually succeeds in practice in learning all types of stress systems. In the sample of stress grammars by Heinz (2009), which the present article will look at, some languages appeared to be entirely unlearnable using Heinz's (2009) parameters, meaning that these stress systems cannot be described using k-testable languages—or, in other words, that Heinz's (2009) learner was not powerful enough. It is important to note, however, that Heinz (2009) looked only at k-testable learning in his article. The present article

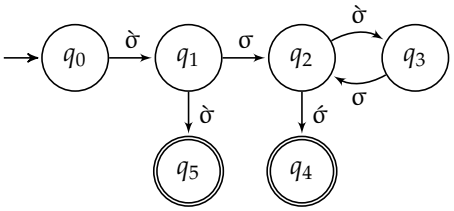

Figure 1
Two-testable finite-state machine based on 'ὸὸ' and 'ὸσὸσό'.

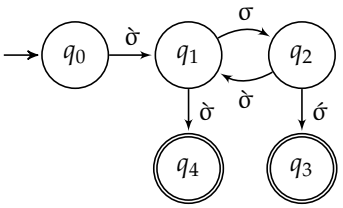

Figure 2
A 1, 1-local finite-state machine based on 'ὸὸ' and 'ὸσὸσό'.

contributes to our knowledge of grammatical induction in the context of stress systems by also considering k,l-local learning (which leads to more fine-grained approaches and includes all k-testable learning approaches), and by investigating the impact of the size of k and l, neither of which steps was taken by Heinz (2009).

3. Learning Algorithm

To test the effects of the k and l parameters on learnability, we constructed and evaluated the effectiveness of a finite-state-machine-based learning algorithm. Starting from a known finite-state machine describing a stress system (which described the target grammar), we first generate example words consisting of stress symbols. This set of example words is the input to the learning system. The result of the learning process is a deterministic finite-state machine that is compared against the original finite-state machine to check whether the same language is learned. Weak equivalence is measured (only the possible output has to be the same for both machines, the actual representational structure may be different). As deterministic finite-state machines can be described using a minimum canonical form, equivalence of the induced language and the target language can be determined in polynomial time.

The learning algorithm requires example words from the target system in order to learn. This example input is created by selecting words from the target machine in a breadth-first manner. The order in which edges that have not yet been selected are selected is random. Using a breadth-first search over the edges in the target machine guarantees that, with a minimum number of samples, all edges in the target machine are covered. Additional samples, if required, may be generated in a similar manner even

Algorithm 1 Sample generation from a deterministic finite-state machine.

Require: FST = $\langle \Sigma, Q, q_0, F, \delta \rangle$
{Σ is a finite set of input symbols}
{Q is a finite set of states}
{q_0 is the initial state}
{F is a finite set of final states}
{$\delta : (Q \times \Sigma) \times Q$ is the transition function; δ^* is the transitive closure.}
1: $\delta_{to\_cover} \leftarrow \delta$ {Set containing all transitions.}
2: $F_{to\_cover} \leftarrow F$
3: $traces \leftarrow [(\epsilon, q_0)]$
4: **while** $\delta_{to\_cover} \neq \emptyset$ **and** $F_{to\_cover} \neq \emptyset$ **do**
5: $(string, state) \leftarrow$ Remove first element from $traces$
6: **if** $state \in F_{to\_cover}$ **or** ($state \in F$ **and** $\exists q \in Q : q \in \delta^*(q_0, string)$ **and** $q \in \delta_{to\_cover}$) **then**
7: **print** $string$
8: $F_{to\_cover} \leftarrow F_{to\_cover} / \{state\}$
9: $\delta_{to\_cover} \leftarrow \delta_{to\_cover} / \delta^*(q_0, string)$
10: **end if**
11: **for all** $s \in \Sigma : \exists q \in Q$ **and** $\delta(state, s) = q$ **do**
12: Append $(string + s, \delta(state, s))$ to $traces$
13: **end for**
14: **end while**

after samples covering all edges in the target machine have been found. This leads to samples that gradually have a larger number of symbols.

The sampling procedure is provided in Algorithm 1. We start with asserting (lines 1 and 2) that all transitions and final states still need to be covered (or to be used by generating a word). We then start with the initial state, which can be reached using the empty word (line 3).

Information about paths through the finite-state machine is stored in the form of **traces**. A trace is a partial pass through the finite-state machine and consists of a (partial) word and a state. The partial word indicates the unique path through the finite-state machine to reach the state. Starting from a trace (line 5), we test whether the state is either a final state that still needs to be covered, or is a final state and the path to the state contains transitions that have not been covered yet (line 6). In either of these situations, the word is written to output (line 7; the word will be part of the training data later) and the used final state (line 8) and used transitions (line 9) are removed from the sets of states and transitions that still need to be used. Finally, all possible transitions that can be reached from the current state are added to the set of traces (line 12). This process is continued until all final states have been used to output words and all transitions in the finite-state machine have been used at least once.

Note that this algorithm can be modified slightly to continue generating example words. Two modifications need to be made. First, on line 4, instead of testing whether all final states and transitions have been covered, we could test whether enough words have been generated. Second, on line 6, it is enough to test whether the state is a final state to output the word. The words that are generated by the sample generation algorithm serve as input to the learning algorithm.

The learning algorithm relies on a theorem stating that given a prefix tree[1] (a deterministic finite-state machine with a single start state, which for each state in the tree denotes a unique prefix of words in the training data) of a sufficient sample provided as training data, a partitioning of the states can be identified such that when merging all states that are in the same part in the partitioning, the result corresponds to a finite-state machine that is isomorphic to the target machine (Heinz 2007). The difficulty lies in identifying the partitioning of the states in the prefix tree. The decision to place states in the same part in the partitioning is performed according to the notion of locality in our approach.

Initially, the learning algorithm builds a prefix tree given the training data. In the data, the sequences (words) consist of symbols describing syllables. The symbols are in the shape of '`w?.s?`', where the question marks indicate a number ranging from 1–4 (in the case of `w`) or 0–2 (in the case of `s`). The number following `w` indicates the syllable weight, and the number following `s` indicates the stress level (where 0 = no stress, 1 = secondary stress, 2 = primary stress).

As an example, consider Figure 3, which describes the stress patterns of Pintupi. Based on the training sample in Figure 4, the corresponding prefix tree can be constructed. This training data and prefix tree are taken from Heinz, de la Higuera, and van Zaanen (2015).[2]

Next, the algorithm searches for states that share a context according to the locality criterion and places them in the same part of the partitioning, as these states

1 A prefix tree is used, even though some languages, such as Dutch [Gussenhoven 2009, 2014], must be described as assigning stress starting from the *right* word edge. Nonetheless, our use of a prefix tree is consistent with the fact that our representation describes time in a left-to-right fashion.

2 Note that the training data contain two more words than are generated by Algorithm 1.

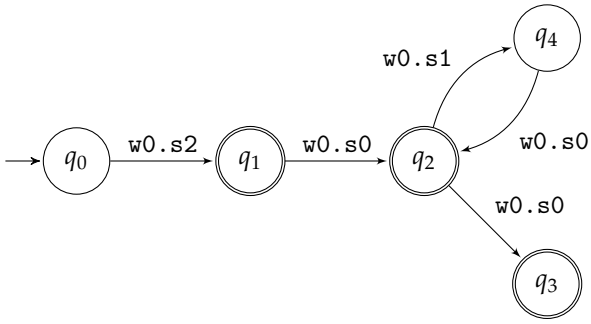

Figure 3
Finite-state machine describing Pintupi stress patterns.


```
w0.s2
w0.s2 w0.s0
w0.s2 w0.s0 w0.s0
w0.s2 w0.s0 w0.s1 w0.s0
w0.s2 w0.s0 w0.s1 w0.s0 w0.s0
w0.s2 w0.s0 w0.s1 w0.s0 w0.s1 w0.s0
```

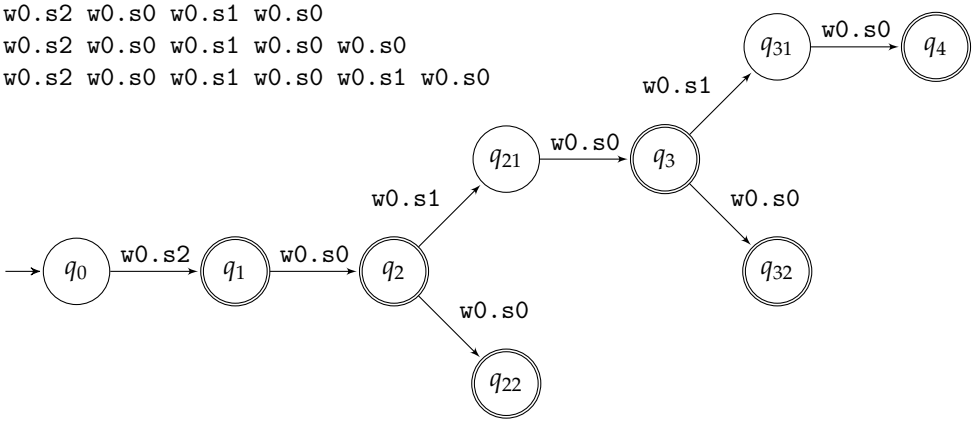

Figure 4
Training data from the finite-state machine for Pintupi stress patterns in Figure 3 and the corresponding prefix tree.

may be merged. This results in a partitioning of the states in the prefix tree. Merging according to this partitioning leads to the final, learned finite-state machine. Figure 5 illustrates the resulting finite-state machine after merging the partitioning based on the two-testable criterion, which leads to a partitioning of the states $[\{q_0\}, \{q_1\}, \{q_2\}, \{q_{21}, q_{31}\}, \{q_{22}, q_{32}\}, \{q_3, q_4\}]$ in Figure 4 into states $[q_0, q_1, q_2, q_3, q_4, q_5]$, respectively, in Figure 5. Note that the resulting finite-state machine (in Figure 5), even though it has a different structure, describes the same language as the finite-state machine in Figure 3. Hence, this language can be learned effectively with the two-testable locality criterion.

The locality criterion is the key factor that determines whether two states are placed in the same part in the partition. When they are, the states will be merged. This criterion is coded by means of two variables: k, representing the length of the sequence of

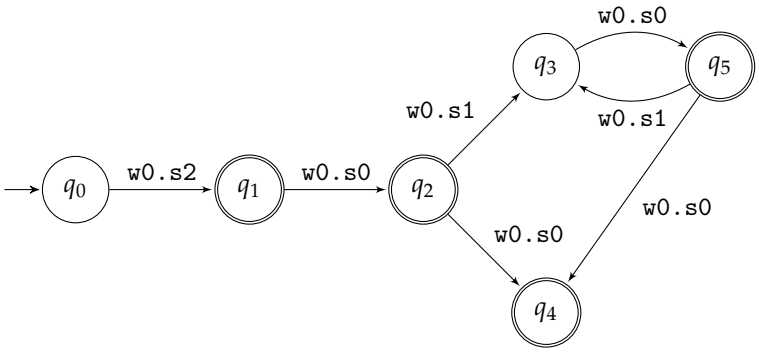

Figure 5
The result of merging states in Figure 4 with the same incoming paths of length 2.

incoming edges with the same symbols for both states (describing left context of the state under consideration for merging) and l, representing the length of the sequence of outgoing edges with the same symbol for both states (describing its right context in the prefix tree). In other words, two states are placed in the same part of the partitioning if

Algorithm 2 Learning algorithm incorporating k-testable and k, l-local merging.

Require: FST = $\langle \Sigma, Q, q_0, F, \delta \rangle$
Require: $k, l \in \mathbb{N}$ {k, l-local, or k-testable if $l = 0$}
{FST is a prefix tree}
{Σ is a finite set of input symbols}
{Q is a finite set of states}
{q_0 is the initial state}
{F is a finite set of final states}
{$\delta : (Q \times \Sigma) \times Q$ is the transition function; δ^* is the transitive closure.}
1: $\pi \leftarrow \emptyset$ {Partition}
2: $Q_{to\_part} \leftarrow Q$
3: **while** $Q_{to\_part} \neq \emptyset$ **do**
4: $q \leftarrow$ Remove state from Q_{to_part}
5: $block = \{q' | \exists x_k, y_k, x_l, y_l \in Q, w_k, w_l \in \Sigma^* :$
$|w_k| = k, \delta^*(x_k, w_k) = q, \delta^*(y_k, w_k) = q', |w_l| = l, \delta^*(q, w_l) = x_l, \delta^*(q', w_l) = y_l\}$
6: $Q_{to\_part} \leftarrow Q_{to\_part}/block$
7: $\pi \leftarrow \pi \cup \{block\}$
8: **end while**
{Merge states in FST}
9: $Q' \leftarrow \pi$ {states are the *blocks* of π}
10: $q'_0 \leftarrow B \in \pi : q_0 \in B$
11: $F' \leftarrow \{B \in \pi : F \cap B \neq \emptyset\}$
12: **for all** $B \in \pi$ **and** $a \in \Sigma$ **do**
13: $\delta'(B, a) \leftarrow \{B' \in \pi : \exists q \in B,\ q' \in B'$ such that $q' \in \delta(q, a)\}$
14: **end for**
15: **return** $\langle Q', q'_0, F', \delta' \rangle$

a sequence of k edges to the left (toward the start state) and a sequence of l edges to the right (following outgoing edges) can be found.[3]

The learning algorithm is described in more detail in Algorithm 2. Starting out with a prefix tree (although any finite-state machine could be used) and values for k and l describing the size of the left and right context, an empty partitioning is defined (line 1) and all states still need to be placed in the partitioning (line 2). Taking a state q that still needs to be placed in the partitioning (line 4), we identify its block. The block contains all states that share the same context as q. Here, w_k and w_l are sequences of length k and l, respectively, that describe the left and right context of the states q and q'. If such a context can be found (by identifying the corresponding states x_k, y_k, and x_l and y_l in the set of states Q), then the states q and q' should be in the same block (line 5). Identifying blocks of states continues until all states have been placed in a block (line 3, with bookkeeping on lines 6 and 7). Once the partitioning has been identified, lines 9–14 create a new finite-state machine according to the partitioning, which is returned in line 15.

We emphasize that with the present approach to learning in general and state-merging specifically, it has been shown, as mentioned earlier, that all k-testable languages are efficiently learnable (Garcia and Vidal 1990). Informally, we know k-testable languages are efficiently learnable, because there exists a partition of the states inside a prefix tree that allows for the efficient identification of the proper states to be merged so as to correctly reconstruct the original finite-state machine that generated the training data (Myhill 1957; Nerode 1958). The partitioning criterion of k-testable languages corresponds exactly to the comparison of the left contexts of the states under consideration.

4. Method

Our work builds on earlier work by Heinz (2009), in no small part because it is based on the same data set.[4] We innovate upon Heinz's (2009) original methodology by including the aforementioned two variations with respect to (i) directionality and (ii) length of the locality window.

The available data comprised 106 transducers, of which 73 modeled actual languages (the remaining 33 modeled artificial stress systems and were excluded following a comment from a reviewer). Given a sequence of syllables (and their weights), these transducers generated sequences of syllables including stress information.

For each transducer in the data set, we create a finite-state machine that describes the stress system. Essentially, a finite-state machine (a finite-state acceptor) describing the output of the transducer is created (via projection, followed by determinization and dead-state elimination). We call this finite-state machine the **target**. From here on, we assume that the data set used in this research consists of 73 target finite-state machines.

In the experiments described herein, we varied values for k and l between 0 and 10 (inclusive on both sides). This means that, in determining whether a state s_1 should be merged with another state s_2, our algorithm compared symbols on the edges of the k states preceding both s_1 and s_2 and also at the symbols on edges of l states following s_1 and s_2. When evaluating two or more states for merging, the symbols on the k preceding

3 There is a discrepancy between the left and right contexts. Because we search through a prefix tree, for each state, there is a unique path toward the start state, but there may be several outgoing paths for each state. States are merged if their incoming paths (of length k) are the same and if there exists at least one outgoing path (of length l) that shares the same symbols for both states.

4 We thank Jeffrey Heinz for making his machine-readable data available to us.

edges must be the same for all of the states under evaluation, and, for each state under consideration, there must be at least one path of l states that has the same symbols on the edges.

Each point of data generated thus consists of a combination of target grammar, k/l settings, the amount of example data presented, and the outcome of the algorithm. This outcome was coded by binary score: Either the induced grammar was equivalent to the target grammar (and hence learning was successful), or it was not. Note that when all possible combinations of k and l are taken into account when evaluating the outcome, we may expect a disproportionally high number of failures compared with the number of successes. The reason is that, for some languages earlier than for others, certain values for k and l will exceed the possibilities that the original finite-state machine can offer. As an example, with $k = 10$ the algorithm will (obviously) fail to merge any states for any word not consisting of at least ten syllables, which will lead to a predictable failure to reconstruct the original finite-state machine in all its parsimony. We controlled for this problematic but avoidable effect by excluding all generated data points fitting the following criteria:

- the learning of the particular language with the provided settings for k and l had not succeeded; and
- there were no higher values for k and l for which this case *did* succeed; and
- there was no other case in which the learning *did* succeed with the provided settings for k and l.

In other words, we tried to heuristically identify the maximum value, or **limit**, for k and l that allowed at least one language to be learned. We explicitly did not establish individual limits per language, since, although it is extremely likely that different languages will start to fail at different k and l values, we seek to model the human learner's *general capacity* for learning stress grammars, rather than seeking to identify limits on individual languages; thus, we explicitly opted to remain language-agnostic in our pruning of the data.

Besides the obvious matter that our requests for sizes of k and l need to respect what can be offered by the languages, the feasibility of the task of grammatical induction also hinges on two additional factors. First of all, some languages may be more difficult to learn than others. Heinz (2009) even notes that some of the languages in this sample are plainly unlearnable with the learning algorithm he used (which is slightly different from ours, as mentioned earlier). As we will explain subsequently, we controlled for possible language-intrinsic differences of this kind in the statistical analysis of the results. A second factor that should obviously influence the success chances of inducing a grammar is the amount of example data we present to the learner. We manipulated this factor by providing between 10 and 500 example words, varied in steps of 10 for the 0–100 interval and in steps of 50 for the 100–500 interval. These example words were generated from the original finite-state machine in such a way that they were maximally representative of the "true" grammar, meaning that they exercised as many edges from the original finite-state machine in the most economic way possible (implemented by disfavoring words containing multiple, redundant, instances of the same part of the stress pattern using breadth-first selection). This was implemented using the modification of Algorithm 1 as described earlier.

A consequence of our breadth-first approach to sampling is that every example is guaranteed to provide a genuine contribution to the learning process as it unfolds. This

allows us to provide the learner with relatively few examples, while still remaining ecologically valid: Although it is true the human learner hears thousands of examples during infancy, most of these will only provide redundant information that was already available in earlier examples. Our learner, on the other hand, can be expected to converge on a stable target grammar much faster than a human learner does, because our sampling method ensures that the earliest examples are already maximally representative. Because our learner is deterministic, the fact that this approach to sampling is not in line with human ontogeny is not an issue: Had we sampled fully randomly rather than breadth-first, it would only have taken the learner longer (i.e., more data) to achieve the same end state. Note that this does not mean that providing more data than minimally required is completely pointless, because the gold-standard finite-state machines might contain loops, in which case more data is required for the learner to identify the placement of the loops in the finite-state machine.

5. Results

At first glance, the results of our learning algorithm look to be somewhat disappointing. Of the 132,495 valid data points collected, only 6,209 data points are successes; the remaining 126,286 represent failures of the algorithm to correctly induce a grammar equivalent to that of the original finite-state machine. When we plot the number of successes against the parameters provided to the model, however, an intriguing picture emerges (Figure 6): It appears that only one specific combination of k and l, namely, $k = 3, l = 0$ really works (although closely followed by $k = 2, l = 0$ and $k = 4, l = 0$). It thus appears that the $k = 3, l = 0$ learner is the best learner in the set tested. This learner was able to learn 471 data points out of the 1,095 data points that had been offered to it (and remained after the pruning step).

The statistical significance of this finding was evaluated by means of forward-stepwise logistic regression analysis. The dependent variable in the model was learnability, which encoded whether the learner had succeeded in reconstructing the original finite-state machine (or an equivalent that generates the same language) or not. In addition to the intercept, two fixed-effect predictors were included: *data* and *learner settings*. The *data* predictor, encoding the number of example words presented to the learner, was log-transformed to make its distribution more normal (thus compensating for our discontinuous sampling of the data) and centered. The *learner settings* predictor was a dummy variable, encoding whether the data were obtained from the $k = 3, l = 0$ learner (coded as 1) or not (coded as 0). To account for the blocking effect of language (samples taken from the same language being likely to be correlated), all fixed effects were included in the model together with the corresponding random slope by language (cf. Barr et al. 2013). All models also included a random intercept by language.

Model comparisons showed that the *data* predictor significantly improved model fit ($\chi^2_1 = 1,106.03$, $p < 0.001$). After controlling for this effect, the *learner type* predictor additionally contributed significantly to the fit of the model: $\chi^2_1 = 2,422.79$, $p < 0.001$. The regression coefficients (provided in Table 1) confirm that all fixed-effect predictors are significant. The intercept shows that, for the non-$k = 3, l = 0$ learners provided with the mean log amount of data, the odds of successful learning are abysmal (OR = 1:2,688.34, $p < 0.001$). The chances of success increase, on average, 5-fold when more data is provided to the learner (OR = 5.35:1, $p < 0.001$). Note that this predictor had been log-transformed, meaning that this 5.35-fold increase holds for every $\log n$ additional words provided; the effect in the original data is plotted in Figure 7 for the $k = 3, l = 0$ learner. Finally, the predictor encoding whether the optimal learner settings were used

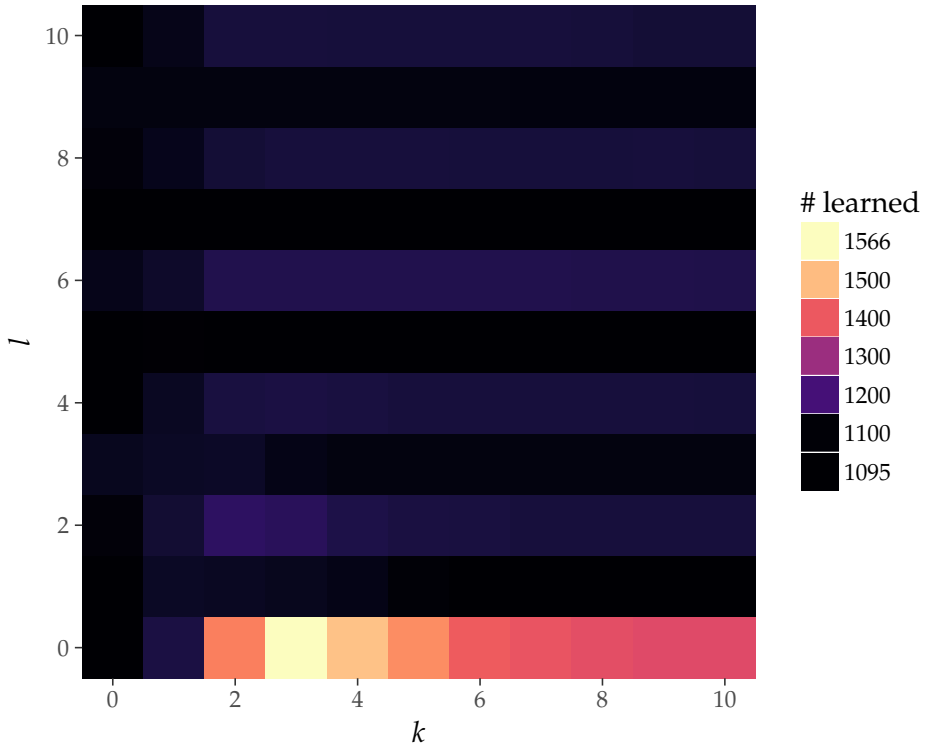

Figure 6
The number of learning successes (summed over the different amounts of example data) for each of the different values for k and l.

Table 1
Fixed-effects estimates for the logistic regression model.

Factor	*Estimate (SE)*	*Odds Ratio*	*z*	*p*
Intercept	−7.88 (0.58)	1:2,688.34	−13.70	<0.001
Log amount of example data	1.68 (0.20)	5.35:1	8.25	<0.001
Learner settings: $k = 3, l = 0$	5.24 (0.28)	189.46:1	18.87	<0.001

showed a significant effect in the expected direction: The learner set to $k = 3, l = 0$ performed 189.46 times as good as the average of the learners provided with different settings; in other words, the $k = 3, l = 0$ learner was significantly better than the others.

Figure 8 shows caterpillar plots for the parameter estimates for the random effects. The estimates are overall not surprising: Some languages turn out to be easy to learn in any case (e.g., Maranungku, where 819 of 1,819 non-pruned settings and data combinations succeeded), some languages appear simply impossible to learn (e.g., Pirahã, where none of the 1,815 non-pruned combinations succeeded). These findings are discussed further in Section 6.

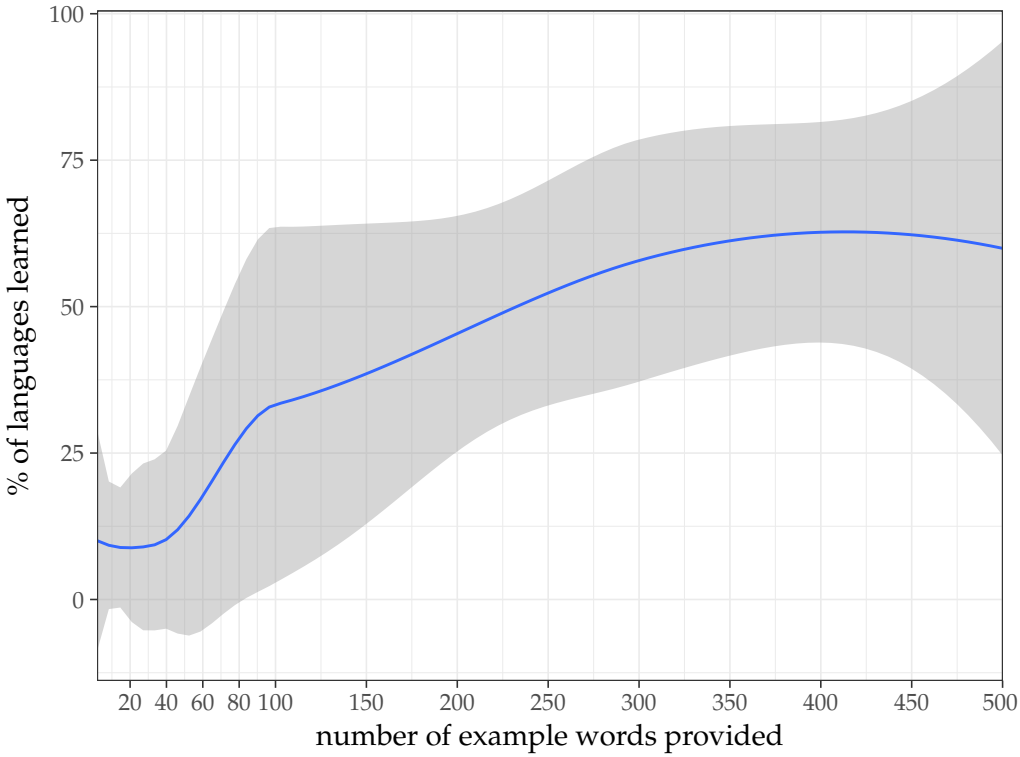

Figure 7
The effect of providing more example words to the $k = 3, l = 0$ learner, averaged over the 73 languages. Because there is significant variability between the languages (cf. Figure 8), the data are less-smoothed for clarity; the shaded area denotes the SE of the smooth.

6. Discussion

The results of both analyses showed, first of all, that there was significant language-to-language variability in the base odds of successful learning. This is fully in line with Heinz (2009), who shows that some languages displaying certain complex properties are completely unlearnable by the algorithm he used. Our analysis replicates this result: There was considerable variation in the random-effects estimates, especially in the intercept, which represented the base odds of success (cf. Figure 8).

As an example, consider the finite-state machine in Figure 9, which describes the stress patterns of Lithuanian. This language is interesting, because it is one of the 25 languages that, according to Figure 8, could not be learned at all. If we apply Algorithm 1 to this language's finite-state machine, we get the sample words as found in Figure 10. In this prefix tree, no merging will occur when using a k-testable merging criterion when $k > 1$. When using a 1-testable criterion, however, the algorithm will generate a partitioning $\{\{q_0\}, \{q_1, q_5\}, \{q_2, q_3\}, \{q_4\}\}$ of states in Figure 10 to states $\{q_0, q_1, q_2, q_3\}$ in the finite-state machine found in Figure 11. The resulting finite-state machine allows for multiple occurrences of `w1.s2` in the words to be accepted, which is not allowed in the original finite-state machine in Figure 9.

Note also that this problem cannot be solved with additional training data or with larger k settings. With $k = 1$, the same states will be merged regardless of the amount of training data, but even with increasing k (and the amount of training data), at some

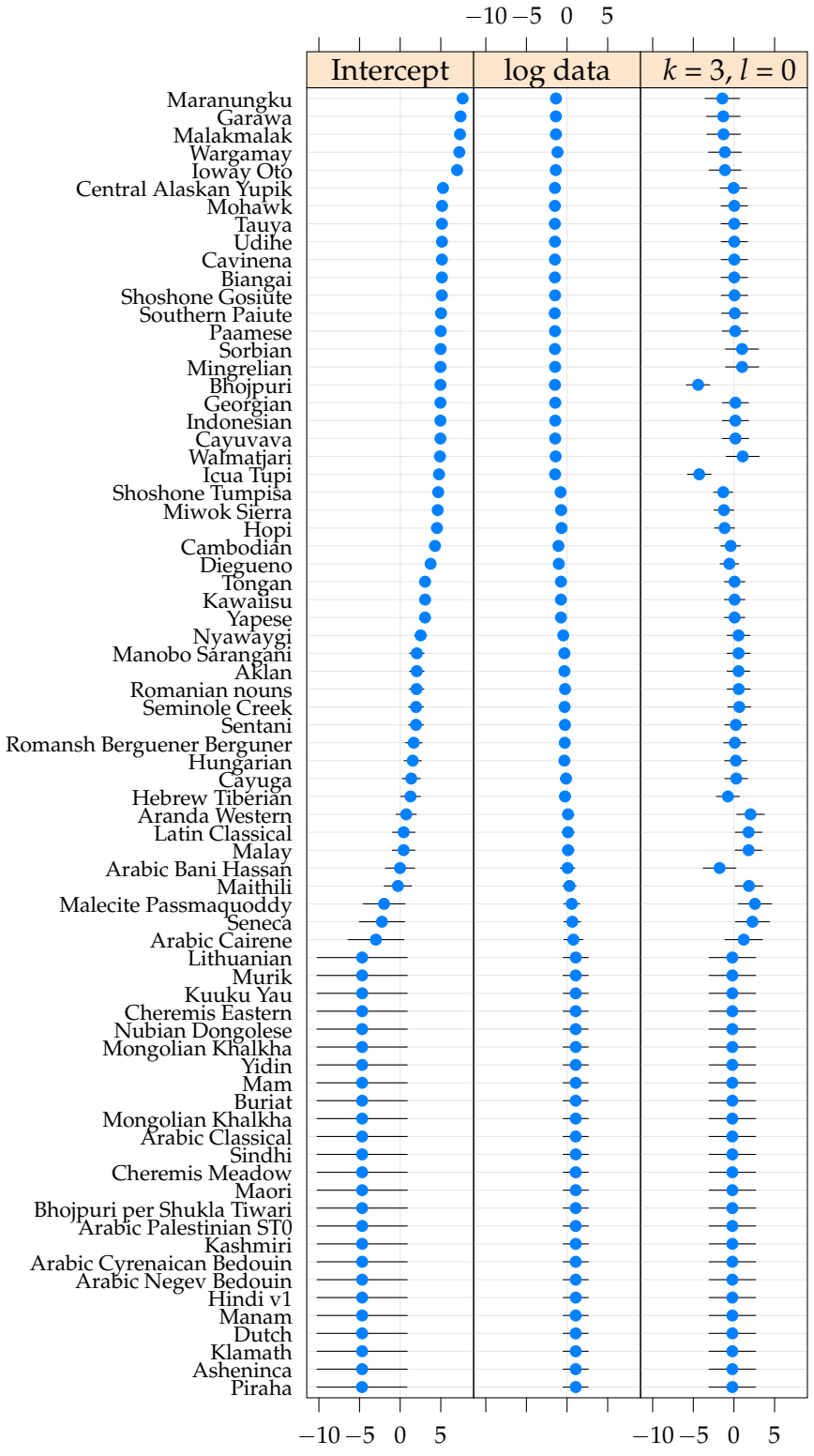

Figure 8
Caterpillar plots showing the estimated random effects by language and their confidence intervals for each of the three terms in the logistic regression model.

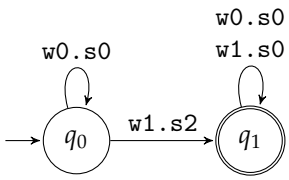

Figure 9
Finite-state machine describing Lithuanian stress patterns.

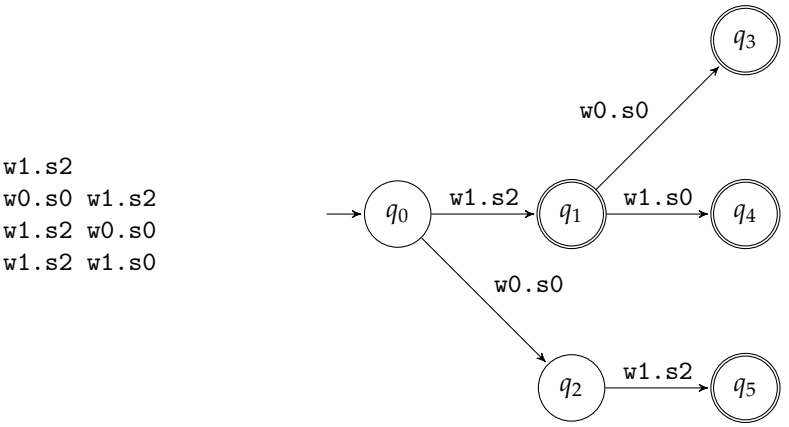

Figure 10
Training data generated by Algorithm 1 from the finite-state machine in Figure 9 and its corresponding prefix tree.

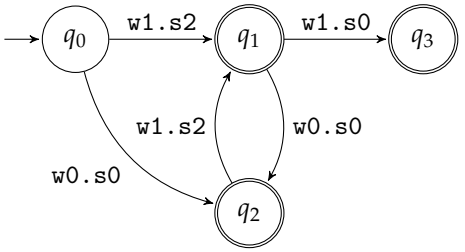

Figure 11
Resulting finite-state machine after 1-testable merging of the prefix tree of Figure 10.

point the learner will not be able to distinguish between the sequences generated by the loop of q_0 and those of the (`w0.s0`) loop of q_1. This means that the resulting finite-state machine loses the ability to keep track of whether `w1.s2` has already been handled. To learn these finite-state machines, the notion of "memory" needs to be maintained and for this, arbitrarily large contexts are required. Hence, this language does not conform to Kenstowicz's (1994) notion of phonological locality.

This idea of memory is interesting. Finite-state machines are only capable of keeping track of a finite amount of memory (bounded ultimately by the number of states in the machine). One may expect that stronger grammatical formalisms can be used to solve the problem of learning languages that defy the notion of phonological locality. For instance, context-free grammars, which correspond to push-down automata, have

access to unlimited amounts of memory. However, they are known not to be efficiently learnable. Furthermore, the problem is not in the representation, as the languages in our data set are represented as finite-state machines, but during learning the algorithm requires more memory, or context, than the algorithms described in this article have available. A more detailed linguistic analysis of the stress patterns that are not learned correctly by the current approach is needed. An initial, informal qualitative analysis shows that non-locality is problematic for the current learning approaches.

Turning to our main manipulation of interest—the effect of varying k and l, permitting us to distinguish between k-testable and k, l-local learning with various sizes for k and l—we observe that taking into account more syllables when considering whether two states in a pattern should really be considered identical (i.e., taking a larger analysis window) yields better results, up to a point: As k increases up to a value of 3, so do the chances of successful learning (Figure 6). This also follows from Heinz (2009), who notes that in all instances in which his learning algorithm failed, it did so because it overgeneralized, which means that the learning algorithm was too eager to merge two at-a-first-glance similar-looking states. The main idea behind restricting the merging of states not only based on a left context, but also on a right context (as investigated in the current research), hinges on this problem.

Based on previous results, we expected that more conservative approaches to state-merging would yield better results. This can be seen from the chart of the successes in Figure 6, where the chances of success are highest at $k = 3$ and are also fairly high for larger ks, although the real hotspot is at $k = 3$. It is interesting to note that this plateauing at $k = 3$ is fully in line with the classic formulation of phonological locality that assumes that each constituent may inspect its two direct siblings only (Kenstowicz 1994), which yields this analysis window of size 3. When k is increased beyond 3, the learnability starts to slowly but gradually decline again, which makes sense: The more context has to be identical for two states to be considered for merging, the harder it becomes to satisfy this criterion; insufficient state-merging then logically leads to a failure to (re)construct a suitably parsimonious grammar. In other words, merging with $k \geq 3$ leads to too strict merging criteria. The sweet spot is at $k = 3$.

Trying to improve on the results with $k = 3$, we expected that adding a (limited size) right context, which is performed using k, l-local merging, would provide a gentle way of restricting state-merging (with respect to k-testable with the same k values). However, this turns out not to be the case.

In this context, taking another look at the actual effect of the merging criterion may provide further insight. In the case of a k-testable merging criterion, two states are merged when they are reachable after analyzing k syllables (the same k syllables for both states). In the case of k, l-local (with $k + l$ the same as the k in the k-testable case), two differences can be identified. Firstly, different states are merged. This is illustrated in Figure 12. In the case of the 3-testable merging criterion, state q_3 is merged with other states (like q_3') that show the same incoming path of three transitions (`a b c`). However, in the 2, 1-local merging criterion case, state q_2 is merged with other states (like q_2') that also have the same incoming path of two transitions (`a b`) and an outgoing transition (`c`). Essentially, 2, 1-local merging corresponds more closely to a 2-testable condition with an additional requirement (a corresponding outgoing transition). Secondly, the k, l-local merging criterion cannot merge final states that do not have any outgoing transitions if $l \geq 1$. This shows that different finite-state machines will be learned (with different generalizations). However, our original aim of defining a less greedy merging approach is achieved: A k, l-local merging approach is less greedy with respect to a k-testable approach with both k values the same and $l \geq 1$. However,

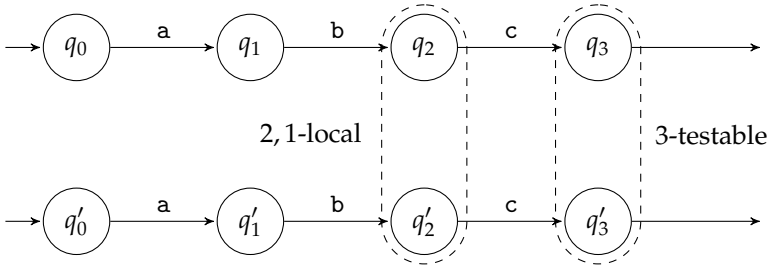

Figure 12
Merging of states based on 3-testable and 2, 1-local merging criteria.

this more cautious strategy does not seem to help in identifying the correct states to be merged.

This illustrates another major finding of the present study: The only definition of "locality" that is really helpful to the learner is one proceeding strictly from the left edge of the state under consideration for merging. We arrive at this conclusion because increasing the value of l, which is the rightward context taken into account from the state under consideration for merging onward, decreases the chances of successful learning—in fact, *any* value other than 0 turns out to be detrimental. In other words: The more we look to the left, the better our results turn out (up to a point, viz. $k = 3$), but the odds of successful learning sharply fall the second we start looking to the right (when evaluating if the context of two states is identical and hence if the two states may be merged).

In conclusion, the two questions we ended with in the Introduction (what is the appropriate size and direction of evaluating the phonological neighborhood?) can be answered as follows. As concerns k-testable vs. k, l-local learning, it turned out that k-testable learning is clearly to be preferred over k, l-local learning; the latter even has a detrimental effect on the learnability of the stress systems we presented to our learner. The best results are achieved when using $k = 3, l = 0$: this allowed the algorithm to successfully learn a non-negligible number of languages (471 data points out of 1,095), though this still amounts to only a little less than half of the available systems. This shows that the linguistic notion of locality holds for many languages.

However, our results also corroborate Heinz (2009), in that some languages indeed appear to simply not be learnable using this type of merging strategy, which means that there must be some property other than only the learner's take on locality that makes these languages difficult to learn (cf. Heinz 2009). A more in-depth analysis of properties of the languages that cannot be learned may reveal information that allows for the development of improved merging strategies.

This work leaves open several interesting avenues for future research. In addition to further investigation in better merging conditions (which take into account the notion of memory), we may take a closer look, as suggested by a reviewer, at the amount and especially the quality (in terms of providing disambiguating evidence) of the data provided. Our research starts from a gold-standard finite-state machine and as such does not represent information on the frequency of paths in the finite-state machine used in daily life. Our results showed a strong increase of the odds of successful learning when more data is offered to the learner, but the data we provided was generated breadth-first and was hence artificial. It is questionable whether real-life language acquisition, which we ultimately seek to model, has the same frequency distribution. We recommend that

future work investigate this further: How much data, and of what quality, is necessary for successful learning on the one hand, and tends to actually be available in naturalistic settings on the other, and what does this tell us about the human capacity for learning metrical stress structure? This approach can be extended even further by modifying the grammatical formalism into a probabilistic one. In this case, the task is to learn a probability distribution over Σ^* that indicates likelihood of sequences occurring in a natural context. Evaluation is then not restricted to a binary decision (learned correctly or not), but to a distance between the gold-standard probability distribution and that of the learned grammar.

Our study ties in naturally with other recent work on learner biases as possible explanations for universal patterns in the typology of stress systems. Staubs (2014), for instance, demonstrates how both the representations as well as the type of input data available to the learner provide a natural bias toward more frequent stress systems. In a similar vein, Stanton (2016) shows that the "midpoint pathology" of stress systems (the, theoretically difficult to explain, observation that there are languages that require stress to be located within a fixed window of syllables from the left or the right word edge, but that these two constraints never conspire together to confine stress to the midpoint of a long-enough word) can be explained by such a stress system being very difficult to learn. Our work contributes to this area of research, by noting another observation: Apparently, the stress systems of the languages of the world appear to be easier to learn using k-testable (specifically, 3-testable) learning, than using k, l-local learning; we thus seem to have taken the first steps towards finding another bias present within the learner.

7. Conclusions

Our experiment ultimately showed that k, l-local learning, whereby locality is evaluated bidirectionally, is a worse (rather than a better) learning method than unidirectional k-testable learning is. In addition, we showed that in general learning is optimal when fixing k to the value of 3, which is in line with the classic description of phonological locality by Kenstowicz (1994). The finding that some languages remain unlearnable by our algorithm extends the same result by Heinz (2009), in that it corroborates his point that there are certain properties yet unknown that still elude finite-state-automata-based stress-grammar induction based on a limited context. The present study has investigated in more detail what these factors detrimental to learning might be, since we have managed to identify at least one of them (k, l-local learning, or including the rightward context when evaluating whether two states should be identical in a finite-state machine representation). We have also identified concrete possibilities for future work, such as using alternative merging conditions (incorporating a notion of memory) and learning using stress sequences occurring in more natural distributions.

References

Árnason, Kristján. 1985. Icelandic word stress and metrical phonology. *Studia Linguistica*, 39(2):93–129.

Barr, Dale J., Roger, Levy, Christoph Scheepers, and Harry J. Tily. 2013. Random effects structure for confirmatory hypothesis testing: Keep it maximal. *Journal of Memory and Language*, 68(3):255–278.

Garcia, Pedro and Enrique Vidal. 1990. Inference of k-testable languages in the strict sense and application to syntactic pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(9):920–925.

van Gijn, Erik. 2006. *A grammar of Yurakaré*. Ph.D. thesis, Radboud University Nijmegen.

Goedemans, Rob and Harry van der Hulst. 2013. Weight-sensitive stress. In Matthew S. Dryer and Martin Haspelmath, editors, *The World Atlas of Language Structures Online*, Max Planck Institute for Evolutionary Anthropology, Leipzig.
Gussenhoven, Carlos. 2009. Vowel duration, syllable quantity and stress in Dutch. In K. Hanson and S. Inkelas, editors, *The Nature of the Word. Essays in Honor of Paul Kiparsky*, MIT University Press, Cambridge, MA, pages 181–198.
Gussenhoven, Carlos. 2014. Possible and impossible exceptions in Dutch word stress. In H. van der Hulst, editor, *Word Stress: Theoretical and Typological Issues*, Oxford University Press, Oxford, pages 276–296.
Hansen, Kenneth C. and Lesley E. Hansen. 1969. Pintupi phonology. *Oceanic Linguistics*, 8(2):153–170.
Hayes, Bruce. 1995. *Metrical Stress Theory: Principles and Case Studies*. University of Chicago Press, Chicago, IL.
Heinz, J. 2007. *The Inductive Learning of Phonotactic Patterns*. Ph.D. thesis, University of California, Los Angeles.
Heinz, Jeffrey. 2009. On the role of locality in learning stress patterns. *Phonology*, 26(02):303–351.
Heinz, Jeffrey, Colin de la Higuera, and Menno van Zaanen. 2015. *Grammatical Inference for Computational Linguistics*, Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers.
Kager, René W. J. 1989. *A Metrical Theory of Stress and Destressing in English and Dutch*. Ph.D. thesis, Utrecht University.
Kenstowicz, Michael J. 1994. *Phonology in Generative Grammar*. Blackwell, Cambridge.
Maskikit-Essed, Raechel and Carlos Gussenhoven. 2016. No stress, no pitch accent, no prosodic focus: The case of Ambonese Malay. *Phonology*. pages 1–37.
Myhill, John R. 1957. Finite automata and the representation of events. Technical Report WADD TR-57-624. Wright-Paterson Air Force Base.
Nerode, Anil. 1958. Linear automaton transformations. *Proceedings of the American Mathematical Society*, 9(4):541–544.
Van Oostendorp, Marc. 2012. Quantity and three-syllable window in Dutch word stress. *Lagnuage and Linguistics Compass*, 6(6):343–358.
Stanton, Juliet. 2016. Learnability shapes typology: The case of the midpoint pathology. *Language*, 92(4):753–791.
Staubs, Robert D. 2014. *Computational Modeling of Learning Biases in Stress Typology*. Ph.D. thesis, University of Massachusetts Amherst.