Learning in the Lexical-Grammatical Interface

Tom Armstrong and **Tim Oates**
Department of Computer Science and Electrical Engineering
University of Maryland, Baltimore County, Baltimore, MD 21250
{arm1, oates}@umbc.edu

Abstract

Children are facile at both discovering word boundaries and using those words to build higher-level structures in tandem. Current research treats lexical acquisition and grammar induction as two distinct tasks. Doing so has led to unreasonable assumptions. Existing work in grammar induction presupposes a perfectly segmented, noise-free lexicon, while lexical learning approaches largely ignore how the lexicon is used. This paper combines both tasks in a novel framework for bootstrapping lexical acquisition and grammar induction. We present an algorithm that iteratively learns a lexicon and a grammar for a class of regular languages in polynomial time, and we report experimental results for benchmark languages.

The ease with which children learn to discover boundaries in their environments while building grounded high-level structures belies the complexity and computational challenges of the task. We address these two disparate problems by proposing a bootstrap between lexical and grammatical knowledge. We improve lexical acquisition through the addition of a new dimension of information and remove a common assumption to all grammar induction algorithms.

Learning grammars is often an intractable problem unless concessions are made regarding the input, and having complete knowledge of the language's alphabet is a common assumption. Learning lexicons from noise-free data is also a challenge, and determining lexical items largely becomes a problem of finding a set of structure-less substrings. It is unrealistic from a developmental perspective to expect perfect information from noisy environments (e.g., child language acquisition, robot sensor data), but state-of-the-art approaches to grammatical and lexical learning require it.

This paper explores the utility of including higher-level structural information in the unsupervised learning of a lexicon and removing the requirement that grammar induction algorithms have perfect, segmented input data. We discuss this learning task in terms of what we call the lexical-grammatical interface where the two tasks are bootstrapped together. In this bootstrap, lexical learning is segmenting sequences of categorical data into an inventory of subsequences based upon grammatical information, and grammar induction is taking segmented sequences of categorical data and building generative structures on top of the data. Learning grammars from segmented data is a hard problem, and learning lexicons from noise-free strings is a hard problem. An autonomous learner embedded in an environment must be able to acquire novel words and adapt existing structures. Our ultimate goal is to extend this work to real-valued sensor data where methods must be robust with respect to noise.

Background and Related Work

The seminal work of Gerry Wolff that has developed into the *compression as computing* paradigm (Wolff 1975) inspires this work. Beginning with the MK10 model in the domain of segmenting natural language text into words, Wolff explored the connection between expressivity and information compression. Over the past thirty years, additional work expanded the analysis of letter digram frequencies to grammatical structures (Wolff 1982; Langley & Stromsten 2000; Wolff 2006). Here we detail some of the extensions in the domains of lexical acquisition and grammar induction.

Lexical Acquisition

The approaches most similar to this work treat acquiring a lexicon as an unsupervised learning problem with a simplicity bias. SEQUITUR, a famous extension of the MK10 model, is an algorithm for compressing a single string (Nevill-Manning & Witten 1997). SEQUITUR is also useful for discovering segment boundaries in natural language text, but does not generalize; the algorithm returns a grammatical representation for a single string and can only produce that string. VOTING-EXPERTS takes a less restrictive approach to finding segment boundaries (Cohen, Heeringa, & Adams 2002). Two experts, entropy and frequency, vote for segment boundaries based on predictability of the next substring and occurrences of a substring, respectively. BOOTLEX explicitly builds and rebuilds a lexicon in terms of itself by reparsing the input strings (Batchelder 2002). It requires an optimal length parameter and does not generalize or produce a hierarchy for the input strings. Other comparable algorithms look at the problem from a child language acquisition perspective (Brent 1999; Hammerton 2002).

Other approaches consider higher level linguistic information. Magerman et al. and Brill et al. consider the problem of segmentation while building a hierarchy using part-of-speech (POS) tagged corpora or only the POS tags (Magerman & Marcus 1990; Brill & Marcus 1992). Both approaches look at natural language POS tagged corpora and segment using the generalized mutual information (GMI) criterion and divergence, respectively. The former finds *distituents* (i.e. constituent boundaries) at data n-gram positions by computing GMI at each candidate location. Multiple iterations of the algorithm result in an *n*-ary branching hierarchy. A constituent is denoted by two boundary distituents and no *a priori* distituents between the two. The latter operationalizes the concept of free variation by counting POS tags and recording the words in the surrounding contexts. If a tag and a pair of tags can occur in the same context, then they construct a rule for these tags. From the counts of words and contexts, they generate rules and compute a divergence value for each rule. After searching over rule space, the algorithm returns the set of rules that minimize overall divergence. Both approaches produce hierarchies, but only work on a small number of categories, with known boundaries, and do not generalize.

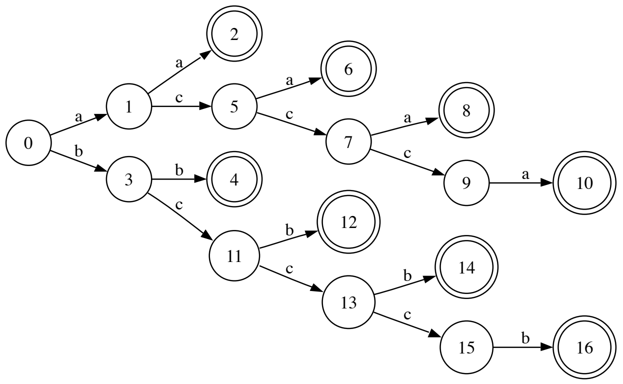

Figure 1: Prefix-Tree Acceptor Built from the Positive Examples {*aa, aca, acca, accca, bb, bcb, bccb, bcccb*}

Grammar Induction

Regular Positive Negative Inference (RPNI) is an algorithm for learning regular grammars from strings in a target language (positive data) and strings not in a target language (negative data) (Carrasco & Oncina 1994). If the input to RPNI consists of a characteristic sample, then the algorithm guarantees learning the target grammar. For the contribution of this paper, it suffices that the algorithm learns the target grammar from data, and we can largely treat the algorithm as a black box (the GI black box in Figure 3). As with other grammar induction algorithms (e.g., ABL (van Zaanen 2000), ADIOS (Solan *et al.* 2005), and AMNESIA (Ron, Singer, & Tishby 1995)), RPNI is not robust with respect to noise (e.g., an improper segmentation of lexical items) and can overgeneralize if the input is not characteristic. However, because RPNI has formal learnability guarantees and is simple to implement, it is a suitable choice.

RPNI is a state-merging algorithm. Initially, it constructs a prefix-tree acceptor (PTA) from the positive data (see Figure 1 built from the strings {*a a, a c a, a c c a, a c c c a, b b, b c b, b c c b, b c c c b*}). States in the PTA are merged if the resulting automaton does not accept any of the negative data. When no more merges can be performed, the resulting machine is guaranteed to accept the target language. Given the PTA from Figure 1 and a set of negative data, RPNI returns the target automata in Figure 4.

Lexical-Grammatical Interface

The lexical-grammatical interface is the interplay between the learning tasks of lexical acquisition and grammar induction (see Figure 3). A typical lexicon learning algorithm begins with a stream of categorical data or a set of strings, and its goal is to induce an inventory of lexical items. A typical grammar induction algorithm begins with a set of strings, and its goal is to learn a generative structural model like the RPNI example above. While lexical learning is done without any regard for structural information, grammar induction assumes a known lexicon and correctly segmented input strings. Motivation for the lexical-grammatical interface comes from child language development. Below, we detail some of the developmental steps that children take that begin with low-level segmentation and then improve the segmentation using higher-level structural information.

Early on in life, children acquire linguistic proficiency in segmenting and parsing. For example, children learn that a prosodic hierarchy governs the structure of speech utterances. The highest level in the hierarchy covers the entire utterance and, in English, starts with high intonation and decreases over the course of the utterance. Inside the utterance, the phonological level governs the structure of a phrase. To assess the development of phonological knowledge in children, Christophe devised a set of sentence pairs where one contained a sequence of phonemes within a phrase and the other crossed a phrase boundary (e.g., an example from the article uses *paper* and *pay per*) (Christophe *et al.* 2003). The experiment found that children at 13 months old preferred the phonological phrase internal presentation of the word. In other words, children as early as 13 months old have acquired sufficient linguistic knowledge about phonological phrase boundaries that they are able to segment fluent speech.

Work by Jusczyk found that children at 7.5 months old that are presented with sparse amounts of data (30 seconds of raw speech audio data) are able to build a representation that persists until at least the following day. At 9 months of age, children already shift their language focus to the typical (i.e., high frequency) phoneme sequences of their native, predominant tongue and spurn words that violate those phonotactic constraints. From 9 to 10.5 months, children demonstrate knowledge of allophonic cues and their use in segmentation in an experiment where "nitrates" and "night rates" are to be distinguished in fluent speech. Later, children begin to generalize over stress patterns and on a high level recognize that word units tend to begin with high stress and end with weak stress (Jusczyk 1996).

Preliminaries

In the remainder of the paper, it is important to maintain the distinction between the alphabet for the set of strings in the lexicon and the alphabet for the finite state automata. Let Σ denote the alphabet for the lexicon and let Γ denote the alphabet for the finite state automata (the lexicon) such that $\Gamma \subseteq \Sigma^*$. Note that in the bootstrap, Σ stays constant in a sample string and Γ varies depending upon the segmentation of the string. At each iteration, Γ adds a new element *uv* such that $u, v \in \Gamma$. Depending upon the remainder of the sample, *u* and *v* may or may not be eliminated from Γ.

Formally, a *deterministic finite automaton* (DFA) is a 6-tuple $\langle Q, \Sigma, \Gamma, q_0, \delta, A\rangle$where Q is a finite set of states, Σ is a finite alphabet, Γ is a finite lexicon of elements in Σ^*, $q_0 \in Q$ is the start state (denoted by 0 in this paper), δ is the function ($\delta : Q \times \Gamma \rightarrow Q$), and $A \subseteq Q$ is a set of accepting states.

Bootstrapping in the Lexical-Grammatical Interface

In this section, we present a novel algorithm that operationalizes the bootstrap for lexical acquisition and grammar induction in the domain of regular grammar learning and string segmentation.

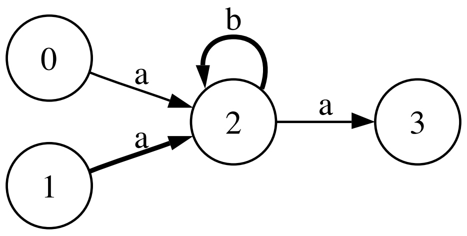

Figure 2: Potential Merge ($1 \xrightarrow{a} 2$ and $2 \xrightarrow{b} 2$)

Figure 3 details the complete structure of the lexical-grammatical interface. The learner receives sequences of data from the environment in the form of phonemic transcriptions. The initial lexicon is the inventory of the phonemes received (a black box segmentation algorithm from above can be used to create an initial segmentation of the sequences). The segmented sequences serve as input to the grammar induction black box and grammar component. The grammar induction black box returns a grammar given in terms of the current lexicon. Up until this point, the process is a standard grammar-induction pipeline. The question is how to use learned grammatical structures to improve the segmentation and, in turn, improve the lexicon.

Consider the way that two rules govern how SEQUITUR compresses its input string. Nonterminals replace pairs of equivalent digrams (digram uniqueness), and each nonterminal must be used more than one time (rule utility). We can leverage the same style of rules when taking the learned grammar and proposing segmentation changes (lexicon refinement). First, we use the learned machine to parse each input string (both positive and negative data). Next, we count the number of strings that pass through each pair of adjacent edges. Frequently traversed pairs of edges in an automaton are frequently occurring digrams in the input strings. However, since we have a higher-level grammatical structure, instead of indiscriminately merging digrams in strings, we have the condition that the same edges must be traversed for a merge to occur.

Edge pair selection proceeds greedily and the most frequent edge pair is used to resegment the positive and negative data (in the case of a tie, the algorithm uses the first edge pair). Figure 2 shows a submachine that highlights a potential merge of the edge labeled a from state 1 to state 2 and the edge labeled *b* from state 2 to itself. Any string that exercises those two productions counts toward the frequency count for the pair of edges. For example, the substring a b b b a beginning at state 1 counts toward the pair frequency once, and if the edge pair is most frequent, then the substring would be resegmented to ab b b a.

The most frequent edge pair drives the resegmentation of both the positive and negative data. The bootstrap continues with the strings in terms of a new lexicon. This complete approach to grammar learning begins with an overly general grammar and proceeds through the space of possible automata to an overly specific grammar (i.e., one production for each string of positive data). The algorithm returns the grammar with the global minimum lexicon size.

Our approach is only a constant amount of time more than the complexity of the choice of grammar induction algorithm. The constant is bounded by the number of possible edge pairs to consider at each step, $\mathrm{O}\big(\Gamma^2\big)$. The number of possible iterations of merges is $\mathrm{O}\big(l * |S_+|\big)$ where l is the length of the longest string in the positive data, S_+. Instead of an exponential blowup in the search space of possible ways to segment the input strings, the bootstrap iteratively improves both the grammar and lexicon with little additional cost.

Experimental Results

We evaluate the bootstrap using natural language data and the state of the art in grammar induction. A perennial problem in both lexical and grammatical learning is proper evaluation. We use data from common corpora with established lexicons and community-derived benchmarks to gauge the success of our approach.

Lexicon Entry	Phoneme Representation
orange	ow r ih n jh
purple	p er p ax l
raisins	r ey z ih n z

Table 1: Sample Lexicon with Phoneme Representation

The alphabet for the lexical learning component is the one and two-letter phoneme representation for word transcriptions called ARPAbet. Words used for the lexicon and the phoneme sequences drawn from the environment are taken from data contained in both the TIMIT and

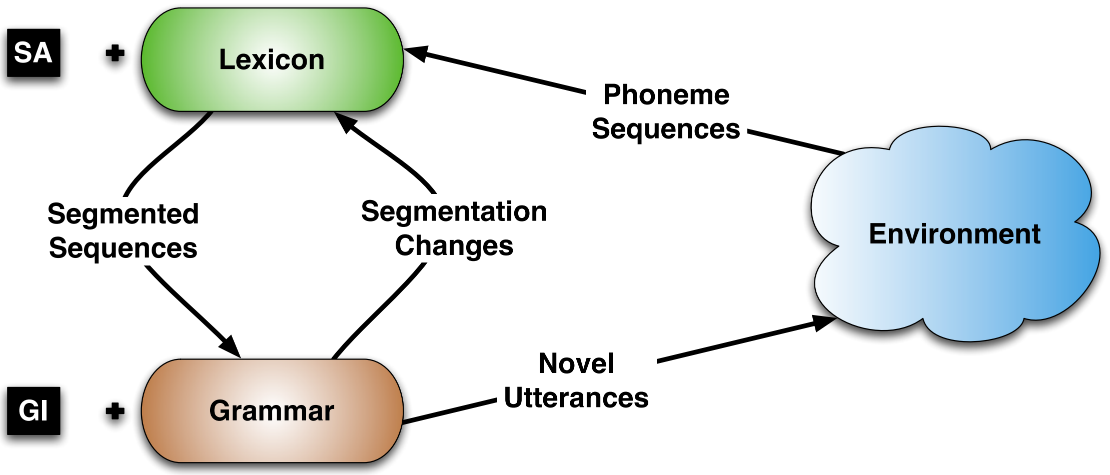

Figure 3: Learning in the Lexical-Grammatical Interface (SA and GI are segmentation and grammar learning black boxes selected based upon the type of sequences and class of languages, respectively)

SWITCHBOARD-1 corpora and their phonemic transcriptions. For example, the word *forward* is transcribed into APRAbet as the following sequence of phonemes: *f ow r w er d*. More example words and transcriptions used in the experiments are available in table 1.

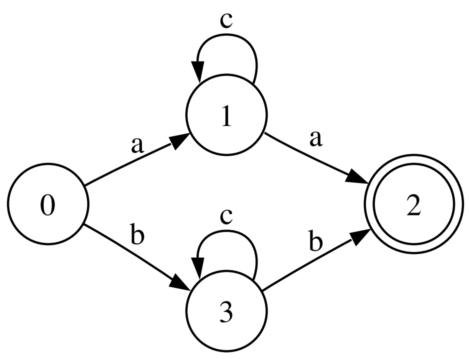

Figure 4: Target Machine Topology

The grammar induction community has a series of benchmark languages for comparing learning algorithms: *L1* through *L15*[1] (Tomita 1982; Dupont 1994). Here we present the results for learning lexicons and grammars for the *L15* benchmark. The algorithm performs analogously on other benchmark languages. The *L15* language contains three lexicon entries and has a regular expression $ac^*a + bc^*b$. In the experiment, *a*, *b*, and *c* are replaced with random ARPAnet phoneme sequences corresponding to words in English.

The automaton in Figure 4 has the target topology for *L15*. Using the three words in the lexicon from table 1, we begin with Σ and $\Gamma = \{\lambda, ow, r, ih, n, jh, p, er, ax, l, ey, z\}$. RPNI constructs a PTA with the positive data, merges states, and finally returns the machine in Figure 5. Therefore, RPNI cannot recover the correct grammar. Also, SEQUITUR does not recover the lexicon given an input consisting of the positive data. While little can be gleaned directly from the learned, overly general automaton, it is able to parse each positive string and will not accept any negative string in table 2. The edge pair starting with the edge labeled *ow* from state 0 to itself followed by the edge labeled *r* from state 0 to itself is the most frequently used to parse the data. The converse edge pair is not counted because it is never used to parse any of the data. The algorithm resegments the lexicon in terms of the winning edge pair and the next iteration begins.

[1]Canonical deterministic finite automata and data are available from `http://www.irisa.fr/symbiose/people/coste/gi_benchs.html`

Positive Data
ow r ih n jh ow r ih n jh
ow r ih n jh r ey z ih n z ow r ih n jh
ow r ih n jh r ey z ih n z r ey z ih n z ow r ih n jh
ow r ih n jh r ey z ih n z r ey z ih n z r ey z ih n z ow r ih n jh
p er p ax l p er p ax l
p er p ax l r ey z ih n z p er p ax l
p er p ax l r ey z ih n z r ey z ih n z p er p ax l
p er p ax l r ey z ih n z r ey z ih n z r ey z ih n z p er p ax l

Negative Data
λ
ow r ih n jh
p er p ax l
r ey z ih n z
ow r ih n jh r ey z ih n z p er p ax l
p er p ax l r ey z ih n z ow r ih n jh

Table 2: Positive and Negative Data for *L15*

Figure 6 shows the size of the lexicon at each iteration in the execution of the algorithm. The algorithm begins with a lexicon of size 12, merges *ow* and *r* to form a new lexicon entry *owr*, but both *ow* and *r* remain in the lexicon. At iteration 19, the algorithm returns the machine with the global minimum lexicon size. Beginning with iteration 20, the learned grammars contain edges that combine two or more correct lexical items. The final iteration, 33, results in a grammar with one edge per positive string. The automaton from iteration 19 in Figure 7 has both the correct structure and the correct lexicon.

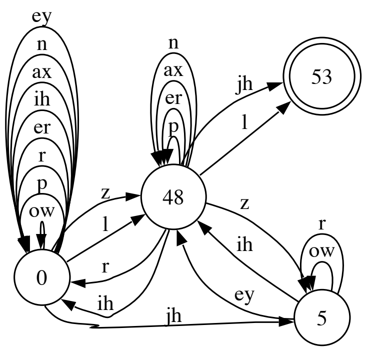

Figure 5: RPNI Result Using the Initial Lexicon

Discussion

For benchmark languages other than *L15* with randomly selected natural language phonemic transcriptions, the algorithm performs analogously. There are, however, cases of languages that present problems. The difficulties with our algorithm and framework result from cases that are challenges for grammar induction algorithms and MK10 and SEQUITUR. For example, SEQUITUR's performance at discovering correct segments increases when the there are more diverse patterns, but degrades when the patterns are more homogeneous.

Consider the pathological case of $\Sigma = \{a\}$, $\Gamma = \{a, aa, aaa\}$, and the language is $(a + aa + aaa)^*$. RPNI requires a small amount of positive and negative data to infer the correct grammar for this language. The positive set of strings is $\{\lambda$, *a, aa, aaa, a a, a aa, a aaa, aa a, aa aa, aa aaa, aaa a, aaa aa, aaa aaa*$\}$ and the negative set of strings is the empty set. There are no strings in Γ^* that are not in the language and there are no strings in Σ^* that are not in the language if we ignore proper segmentation.

For other languages, the algorithm learns a lexicon and a grammar, but overgeneralizes in an interesting way. The general topology of the automaton is correct, but there are additional edges containing one or more concatenations of true lexical items. If we ignore the spacing of strings in the language, the learned grammars will only produce strings in the language. That is, in these cases the overgeneralization does not produce strings that violate the linear order of lexical items in correct strings.

If we use the strings from the positive data above, our algorithm runs into some problems. The lexicon begins as the singleton set $\{a\}$ and the strings *a aa* and *aa a* become indistinguishable. Moreover, strings not in the language (e.g., *aaaa*) are also indistinguishable from strings in the language. These problems are also issues for MK10, SEQUITUR, and VOTING-EXPERTS as they all fail to properly segment the strings.

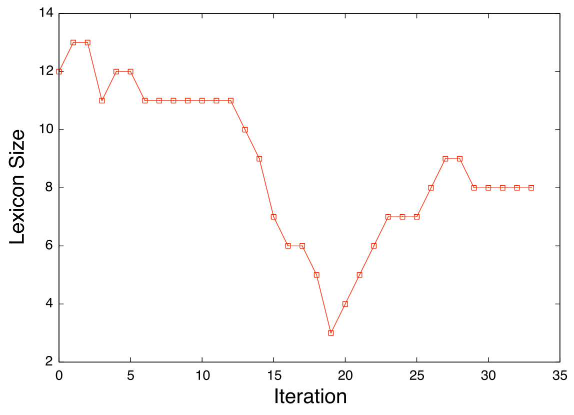

Figure 6: Lexicon Size Over 33 Iterations

Conclusion and Future Work

In this paper, we presented a novel framework for bootstrapping learning a lexicon and inducing a grammar. Prior work on lexical acquisition has ignored valuable grammatical information, and grammar induction approaches require perfectly segmented alphabets to recover correct grammars. We also introduced an efficient algorithm for segmenting strings and learning a grammar for a class of regular languages. Our algorithm begins with an incorrect segmentation of input data and recovers a lexicon and grammar.

Future work will proceed in four directions. First, we will focus on the theoretical boundaries of the lexical-grammatical bootstrap. That is, we will explore the classes of languages that are lexical- and grammatical-learnable in the Chomsky hierarchy. As the grammar induction component is a black box, learning algorithms for more complex languages can replace RPNI. The interested reader can consult Lee's survey article on context-free grammar learning algorithms (Lee 1996). These results will define a new class of learnable formal languages and will shed light on the learnability of natural languages.

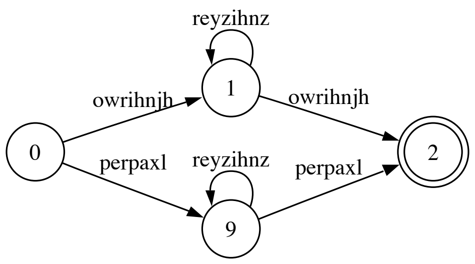

Figure 7: Learned Machine

Next, we will evaluate other measures of success to better understand how to compare grammar and lexicon learning approaches. For example, Büchi automata, finite-state automata for strings of infinite length, have become increasingly popular in the software engineering community. They have developed approximation and simulation techniques that may be useful in experimenting with the generative capacity of inferred grammars (Henzinger, Kupferman, & Rajamani 2002).

Third, we will harness the generative power of the grammar and lexicon to create novel utterances (see Figure 3). A learner embedded in an environment can then be used to experiment with language generation. For example, from the pathological case, the string *a a a* generated by a learned automaton and *aaa* from the target language are both surface equivalent if the segmentation is ignored. This information can be used during edge-merge selection. The intuition is that while the automata's structures are different, the surface forms of the utterances are the same. Finally, we will extend our current results using categorical data to time series data and spoken natural language data.

Acknowledgments

This material is based upon work supported by the National Science Foundation under NSF CAREER award No. 0447435.

References

Batchelder, E. O. 2002. Bootstrapping the lexicon: A computational model of infant speech segmentation. *Cognition* 83(2):167–202.

Brent, M. 1999. An efficient, probabilistically sound algorithm for segmentation and word discovery. *Machine Learning* 34:71–105.

Brill, E., and Marcus, M. 1992. Automatically acquiring phrase structure using distributional analysis. In *HLT '91: Proceedings of the workshop on Speech and Natural Language*, 155–159. Morristown, NJ, USA: Association for Computational Linguistics.

Carrasco, R. C., and Oncina, J. 1994. Learning stochastic regular grammars by means of a state merging method. In *ICGI '94: Proceedings of the Second International Colloquium on Grammatical Inference and Applications*, 139–152. London, UK: Springer-Verlag.

Christophe, A.; Gout, A.; Peperkamp, S.; and Morgan, J. 2003. Discovering words in the continuous speech stream: the role of prosody. *Journal of Phonetics* 31:585–598.

Cohen, P. R.; Heeringa, B.; and Adams, N. M. 2002. Unsupervised segmentation of categorical time series into episodes. In *ICDM*, 99–106.

Dupont, P. 1994. Regular grammatical inference from positive and negative samples by genetic search: the gig method. In *ICGI '94: Proceedings of the Second International Colloquium on Grammatical Inference and Applications*, 236–245. London, UK: Springer-Verlag.

Hammerton, J. 2002. Learning to segment speech with self-organising maps. *Language and Computers* Computational Linguistics in the Netherlands(14):51–64.

Henzinger, T. A.; Kupferman, O.; and Rajamani, S. K. 2002. Fair simulation. *Inf. Comput.* 173(1):64–81.

Jusczyk, P. W. 1996. Investigations of the word segmentation abilities of infants. In *Proceedings of the Fourth International Conference on Spoken Language Processing ICSLP*, volume 3, 1561–1564.

Langley, P., and Stromsten, S. 2000. Learning context-free grammars with a simplicity bias. In *Machine Learning: ECML 2000, 11th European Conference on Machine Learning, Barcelona, Catalonia, Spain, May 31 - June 2, 2000, Proceedings*, volume 1810, 220–228. Springer, Berlin.

Lee, L. 1996. Learning of context-free languages: A survey of the literature. TR TR-12-96, Center for Research in Computing Technology, Harvard University.

Magerman, D. M., and Marcus, M. P. 1990. Parsing a natural language using mutual information statistics. In *AAAI*, 984–989.

Nevill-Manning, C. G., and Witten, I. H. 1997. Identifying hierarchical structure in sequences: A linear-time algorithm. *Journal of Artificial Intelligence Research* 7:67.

Ron, D.; Singer, Y.; and Tishby, N. 1995. On the learnability and usage of acyclic probabilistic finite automata. In *COLT '95: Proceedings of the eighth annual conference on Computational learning theory*, 31–40. New York, NY, USA: ACM Press.

Solan, Z.; Horn, D.; Ruppin, E.; and Edelman, S. 2005. Unsupervised learning of natural languages. *Proc Natl Acad Sci U S A* 102(33):11629–11634.

Tomita, M. 1982. Dynamic construction of finite automata from examples using hill climbing. In *Proceedings of the 4th Annual Cognitive Science Conference*, 105–108.

van Zaanen, M. 2000. ABL: Alignment-Based Learning. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING); Saarbrücken, Germany*, 961–967. Association for Computational Linguistics.

Wolff, J. G. 1975. An algorithm for the segmentation of an artificial language analogue. *British Journal of Psychology* 66:79–90.

Wolff, J. G. 1982. Language acquisition, data compression and generalization. *Language and Communication* 2:57–89.

Wolff, G. J. 2006. *Unifying Computing and Cognition*. CognitionResearch.org.uk.