Abstract

We have developed an efficient algorithm for determining if a Finite State Automaton describes a Strictly Local (SL) stringset, the simplest class of the Sub-Regular Hierarchy, and to determine if that stringset is a subclass of SL that is learnable by an Inductive Inference Machine. We have used this to categorize the phonotactic patterns in a catalog including essentially all of the currently attested patterns occurring in natural languages, most of which turn out to be learnable stringsets in this simplest class.

1 Introduction

Humans are language learners. While it seems simple enough to us, the process of language learning is one of the great mysteries of cognitive science: how do we build up vocabularies (lexicons) and figure out how to combine words and phrases (syntax)? Consider lexicon learning. It cannot be the case that humans master lexicons purely through memorization. As Morris Halle (as quoted in Heinz [4]) points out, English speakers can correctly recognize the English words in the list:

\[
\text{ptak, thole, hlad, plast, sram, mgla, vlas, flitch, dnom, and rtut}
\]

without ever having seen or heard the words before. Thus, human language learners must somehow gain an understanding, without being explicitly taught, of what sound patterns constitute valid words in a given language. The study of such phonological patterns (Phonotactics) is a part of the field of Phonology; this paper is concerned with the intersection of Phonotactics and the Theory of Computation.

Phonologists generally model phonotactic patterns as stringsets and the formalisms they use to describe these stringsets are generally equivalent to Finite State Automata (FSAs). Stringsets described this way are called Regular Stringsets. Regular Stringsets present a problem for learning because they are provably unlearnable by Inductive Inference Machines (IIMs), the simplest formal model of learning [3].

Jeff Heinz treats the problem of learning the lexicon in his dissertation [4]. As part of his work he has created a catalog of nearly all attested phonotactic patterns as well as learning algorithms for each pattern’s representative stringset. The formal properties of the classes
of stringsets learnable by his algorithms are however not yet well understood. We are exploring the patterns in Heinz’s catalog with respect to a well understood hierarchy of classes of stringsets called the Sub-Regular Hierarchy, in which each level corresponds to specific abstract cognitive capabilities and which includes some classes learnable by IIMs. This paper presents the first stage of our exploration, which deals with the simplest class of the hierarchy, the Strictly Local (SL) stringsets [8].

Since the stringsets in Heinz’s catalog are stored as FSAs, we have developed a polynomial time algorithm that can, given an FSA, determine whether or not the stringset it describes is SL. While a polynomial time algorithm, due to Kim, et al., [6] (and implemented by Caron [1]), exists that accomplishes this task, ours is conceptually simpler and easy to implement. This paper makes two contributions. First we present our simplified algorithm, prove its correctness and establish its asymptotic complexity. Secondly, we present the results of applying our this algorithm to the phonotactic patterns of Heinz’s catalog, thus providing a classification of those patterns with respect to the first level of the Sub-Regular Hierarchy.

Section 2 of this paper provides an overview of SL and FSAs, Section 3 documents our algorithm and Section 4 presents our findings about the patterns in Heinz’s catalog. We close with some observations about the cognitive significance of our results and some comments about our ongoing work in this area.

2 Formal Foundations

2.1 Strictly Local Stringsets

An explanation of the Strictly Local (SL) class of stringsets depends on the idea of $k$-factors. A $k$-factor is a sequence of $k$ symbols. The $k$-factors of a string are the sequences of $k$ symbols that appear in that string.

Definition 1. (Set of $k$-factors of a string) The set of $k$-factors of a string $w$ is

$$f_k(w) \overset{\text{def}}{=} \begin{cases} \{v \mid |v| = k \text{ and } w = w_1vw_2 \text{ for some } w_1, w_2\} & \text{if } |w| > k \\ \{w\} & \text{if } |w| \leq k \end{cases}$$

where $|x|$ represents the length of string $x$. For example, let $w = abc$. Then

- $f_1(w) = \{a, b, c\}$
- $f_2(w) = \{ab, bc\}$
- $f_3(w) = f_4(w) = \cdots = \{abc\}$

A stringset is a member of the class SL$_k$ for some natural number $k$ if the stringset contains all and only those strings whose $k$-factors are drawn from a specified list.

Definition 2. (Description of an SL$_k$ stringset) The description of an SL$_k$ stringset is an ordered pair $D = \langle \Sigma, T \rangle$ where $\Sigma$ is an alphabet and $T$ is a set of $k$-factors over $\Sigma \cup \{\times, \kappa\}$. The stringset described by $D$ is denoted $L(D)$.

Definition 3. (Membership in an SL$_k$ stringset) Given a description $D = \langle \Sigma, T \rangle$ of an SL$_k$ stringset, a string $w$ is in $L(D)$ if and only if $f_k(\times w \kappa) \subseteq T$. (Where $\times$ and $\kappa$ are beginning and end markers, respectively.) For example, let

$$M = \langle\{a, b, c\}, \{\times ab, abc, bca, cab, bc\kappa\}\rangle.$$ 

Then

$$L(M) = \{(abc)^i \mid i \geq 1\}.$$

The class of Strictly Local Stringsets contains all stringsets that are SL$_k$ for any natural number $k$. That is

$$\text{SL} \overset{\text{def}}{=} \bigcup_{k \in \mathbb{N}} \text{SL}_k.$$
One way to imagine an $SL_k$ stringset is as one that can be recognized by a machine with a scanning window of width $k$ and a look-up table of $k$-factors. When presented with any string, this scanner will start at the beginning and scan along the string, advancing the position of its window by one symbol at a time so as to examine each $k$-factor. The scanner will accept the string if and only if every $k$-factor that appears in its window over the course of the computation is found in the look-up table.

Any cognitive mechanism that distinguishes an $SL_k$ pattern needs only to be sensitive to the last $k$ phonemes or symbols it has heard. Moreover, if $k$ is known then an $SL_k$ pattern can be learned by an IIM which simply accepts the string if and only if every $k$-factor is known then an $SL_k$ stringset is as

The distinguishing characteristic of $SL$ stringsets is Suffix Substitution Closure.

**Theorem 1 (Suffix Substitution Closure)** A stringset $L$ is $SL$ if and only if there exists some $k \in \mathbb{N}$ such that, for all strings $w_1, v_1, w_2, v_2$, and $x$ such that $|x| = k - 1$,

$$w_1 xv_1 \in L \land w_2 xv_2 \in L \Rightarrow w_1 xv_2 \in L.$$ 

Our definition of $SL_k$ follows the definition in McNaughton and Papert [8]. The Suffix Substitution Closure property is taken to be the definition by McNaughton [7] and Kim, McNaughton and McCloskey [6]. The proof of Theorem 1, that these two are equivalent, is a matter of folklore.

The “only if” is fairly easy to justify; it is essentially another way of stating that an $SL_k$ stringset must contain every string made up of acceptable $k$-factors. If $w_1 xv_1$ is in $L$, then every $k$-factor of $xw_1 x$ is in the table of permitted $k$-factors associated with that stringset. Similarly, if $w_2 xv_2$ is in $L$, then every $k$-factor of $xv_2 x$ is in the table of permitted $k$-factors. This accounts for every $k$-factor of $xw_1 xv_2 x$, meaning that that string must also be in $L$ if the stringset is $SL_k$.

The proof of the other half of Theorem 1 is not quite as immediate. Still, it is not difficult to prove that if $L$ satisfies suffix substitution closure for some $k$, then the union of $f_k(x \cdot w \cdot x)$ over all $w \in L$ is an $SL_k$ description of $L$.

Thus suffix substitution closure characterizes the class of Strictly Local Stringsets.

### 2.2 Finite State Automata

While the notion of an FSA is well known (for instance, see [5]), its formal definition varies. In this paper we use:

**Definition 4.** (FSA) $M = (\Sigma, Q, q_0, \delta, F)$ where:

- $\Sigma$ is the set of symbols in the alphabet
- $Q$ is a finite set of states
- $q_0 \in Q$ is the start state
- $\delta \subseteq Q \times \Sigma \times Q$ is the set of transitions
- $F \subseteq Q$ is the set of accept states

We will interpret this as an edge labeled digraph (the Transition Graph) where $\delta$ is the edge relation.

An FSA is deterministic (a DFA) if no two edges from a single state have the same label, i.e., $\delta$ is a partial function.

**Definition 5.** $M$ is deterministic if $\delta$ is partial functional in its first two places.

If an FSA is deterministic then for every $q_1$ and $w$ there exists at most one $q_2$ such that there is a path labeled $w$ from $q_1$ to $q_2$. For such FSAs we define the path function, $\hat{\delta}(q_1, w)$.

**Definition 6.** (Path function) $\hat{\delta}(q_1, w) \overset{\text{def}}{=} q_2$ where there is a path labeled $w$ from $q_1$ to $q_2$.

A string $w$ is accepted if and only if there is a path labeled $w$ from the start state to an accept state.

**Definition 7.** (Stringset recognized by $M$)

$$L(M) \overset{\text{def}}{=} \{ w \in \Sigma^* \mid \hat{\delta}(q_0, w) \in F \}$$
For any deterministic FSA there is, up to isomorphism, exactly one minimal FSA. For an FSA to be minimal it must have the smallest number of states possible without changing the stringset it accepts. To determine which states are needed to recognize $L$ we utilize Nerode Equivalence with respect $L$, $(\equiv_L)$.

**Definition 8.** (Nerode Equivalence) for $w, v \in \Sigma^*$

$$w \equiv_L v \iff (\forall u)[wu \in L \iff vu \in L]$$

This states that two strings are equivalent relative to some stringset $L$ if and only if there is no string you can append to them that will distinguish them, i.e., extend one to be in the stringset, but not the other.

If two strings are not Nerode equivalent with respect to $L(M)$ then the paths labeled by those strings in $M$ must lead to distinct states. However, if two strings are equivalent then there is no need for the paths they label to end up at distinct states; in a minimal DFA they must end up at the same state. In addition, a minimal DFA may not contain any states that are not reachable from the start state.

**Definition 9.** $M$ is minimal $\iff$

1. $(\forall w_1, w_2 \in \Sigma^*)$

$$[\hat{\delta}(q_0, w_1) = \hat{\delta}(q_0, w_2) \iff w_0 \equiv_L w_1]$$

2. $(\forall q \in Q)([\exists w \in \Sigma^*)[\hat{\delta}(q_0, w) = q]]$

Any state from which there is no path to an accept state is known as a *sink state*. All strings labelling paths that lead to such a state are equivalent with respect to the stringset. This implies that a minimal FSA will only have a single sink state.

Removing the sink state does not affect the stringset the FSA accepts because it does not alter paths that lead to an accept state. It can simplify things to trim away this state.

**Definition 10.** $M$ is trimmed if it’s minimal and has no sink state.

We hereafter assume that all FSAs are deterministic, minimal and trimmed.

### 3 Deciding SL and Finding $k$

#### 3.1 The Basic Algorithm

Our algorithm is based on the following lemma.

**Lemma 1** A stringset recognized by an FSA is SL if and only if there is some $k > 1$ such that for every string $x$ where $|x| = k - 1$, every path in the Transition Graph labeled $x$ leads to the same state.$^1$ If such a $k$ exists, then the stringset is SL$_k$.

The proof is in Section 3.2.

The algorithm, then, is looking for a $k$ for which all paths in the transition graph of the FSA that are labeled with the same string of length $k - 1$ lead to the same state. This is done by simultaneously looking at all paths of length $k - 1$ for progressively greater $k$.

We can do this by building a tree in which the vertices are subsets of $Q$ and the edges are labeled with symbols in $\Sigma$. Each path through the transition graph of length $k - 1$ will correspond to a path from the root of the tree to a vertex at depth $k - 1$.

Since every state is at the end of a path of length 0, the starting vertex is the whole set $Q$. Each vertex $v$ has a successor for each $\sigma \in \Sigma$. Each of these successors is labeled with the set of states that is reachable from some state in $v$ by a transition on $\sigma$.

By Lemma 1, if the stringset is SL$_k$ then the vertices in this tree at depth $k - 1$ will be singletons.

$^1$Although we developed this independently, it is Theorem 2.3 of Caron [1].
function SLk(M)
input : M = (Σ, Q, q₀, δ, F), a trimmed, minimal, deterministic FSA.
output : k ∈ N if \( L(M) \in \text{SL}_k \),
\( ∞ \) otherwise.
construct: \( V = \{ ⟨S, c, h⟩ | S \subseteq Q, c ∈ \{\text{White, Gray, Black}\}, h \in \mathbb{N} \} \)
begin
  v ← ⟨Q, White, 0⟩
  \( V ← \{v\} \)
  SLkSearch(\( M, V, v \))
  return \( \{ ∞ \) if \( \text{HEIGHT}(v) = ∞ \),
  \( \text{HEIGHT}(v) + 2 \) otherwise. \)
end

Figure 1: The Basic Algorithm

Since the cardinality of the set of states labeling the successors of a vertex will be no greater than the cardinality of the set of states labeling that vertex, and since \( Q \) has finitely many subsets, there is either a depth of all singletons, hence \( L(M) \in \text{SL} \), or there is, by the Pigeon Hole Principle, some branch that has a repeated sequence of non-singleton sets and \( L(M) \) is not \( \text{SL}_k \) for any \( k \).

In practice we do not build the whole tree. Instead we build a spanning tree of the corresponding digraph in which there is at most one vertex labeled with each subset. We are, actually, only interested in the subgraph of this graph that is generated by the set of vertices of cardinality greater than 1. This subgraph will be cyclic iff there is repeated sequence of non-singletons along some branch of the tree. Otherwise it will be a rooted directed acyclic graph (DAG).

Definition 11. (Height of a digraph) The height of the digraph rooted at \( v \) is \( ∞ \) if the graph is cyclic otherwise it is the length of the longest path from \( v \) to a leaf.

The successors, in the full graph, of the vertices that have no successors in the subgraph will be at most singleton. Thus, if the subgraph is acyclic and has height \( h \), the vertices at depth \( h + 1 \) in the original tree are at most singletons. By Lemma 1, then, \( L(M) \) is \( \text{SL}_{h+2} \). Conversely, if \( L(M) \) is \( \text{SL}_k \) for some \( k \), then the vertices at depth \( k - 1 \) in the original tree are at most singleton and the height of the DAG is no more than \( k - 2 \).

We implement this as an ordinary depth first graph search [2]. We create the non-singleton/non-empty vertices as we find them, color them white and give them height 0. When we search a vertex we turn it from white to gray, and then recur with any white successors that it has. When we have visited all the successors of a vertex we turn it black. If a vertex has a gray successor then there is a cycle and the height is infinite. Otherwise, the height of a vertex is either \( 0 \), if it has no successors, or it is one plus the maximum of the height of its successors.

3.2 Partial Correctness

We prove partial correctness in four steps. First, we prove the lemma on which the algorithm is based.

Proof (of Lemma 1): Let \( M \) be a deterministic, minimal, trimmed FSA.
function SLkSearch \((M, V, v)\)
input : \(M = (\Sigma, Q, q_0, \delta, F)\), a trimmed, minimal, deterministic FSA.
input : \(V = \{ (S \in \mathcal{P}(Q), c \in \{White, Gray, Black\}, h \in \mathbb{N}) \}\) the set of nodes created so far
input : \(v = (S, c, h) \in V\)
construct: \(V\) is updated with any newly found nodes
Let:
COLOR\((v) = c,\)
HEIGHT\((v) = h\)
begin
COLOR \((v) \leftarrow Gray\)
foreach \(\sigma \in \Sigma\) do
\(S_c \leftarrow \{q_2 \in Q \mid (\exists q_1 \in S) [\hat{\delta}(q_1, \sigma) = q_2]\}\)
if \(\text{card}(S_c) \geq 2\) then
if \(\langle S_c, c_c, h_c \rangle \not\in V\) then
\(V \leftarrow V \cup \{\langle S_c, White, 0 \rangle\}\)
end
\(w \leftarrow \langle S_c, c_c, h_c \rangle\)
if COLOR \((w) = Gray\) then
\(\langle v, w \rangle \) is a back edge.
HEIGHT \((v) \leftarrow \infty\)
else
if COLOR \((w) = White\) then
\(\langle v, w \rangle \) is a tree edge.
SLkSearch \((fsa, V, w)\)
end
\(\langle v, w \rangle \) is either a tree or cross edge.
HEIGHT \((v) \leftarrow \)
\[
\begin{cases} 
\infty & \text{if HEIGHT}(w) = \infty, \\
\max(\text{HEIGHT}(v), \text{HEIGHT}(w) + 1) & \text{otherwise}.
\end{cases}
\]
end
end
(Finished exploring subgraph rooted at \(v\))
COLOR \((v) \leftarrow Black\)
end

Figure 2: The Search Algorithm
Suppose that there exists some $k > 1$ such that for every string $x$ satisfying $|x| = k - 1$, every path labeled $x$ in the transition graph of $M$ leads to the same state.

To show that $L(M)$ satisfies suffix substitution closure for $k$ and is thus $SL_k$, suppose that, for some $w_1, w_2, v_1, v_2$, and $x$ such that $|x| = k - 1$,

$$w_1xv_1 \in L(M) \text{ and } w_2xv_2 \in L(M).$$

Then both $\delta(q_0, w_1xv_1)$ and $\delta(q_0, w_2xv_2)$ are in $F$.

Now, by the first assumption, there is some state $q_x$ such that

$$\hat{\delta}(\delta(q_0, w_1), x) = \hat{\delta}(\delta(q_0, w_2), x) = q_x.$$

By our second assumption, $\hat{\delta}(q_x, v_1) \in F$, hence $\delta(q_0, w_1xv_2) \in F$ and $w_1xv_2 \in L(M)$.

It follows by Theorem 1 that $L(M) \in SL_k$ and, thence, that the stringset $L(M)$ is in $SL$.

$(\Rightarrow)$ (by contraposition) Suppose that there exists no $k > 1$ such that for every string $x$ satisfying $|x| = k - 1$, every path labeled $x$ in the transition graph of $M$ leads to the same state.

Then for all $k > 1$ there exists a string $x$ satisfying $|x| = k - 1$ such that for some states $q_1, q_2, q_{x_1}, q_{x_2} \in Q$,

$$\hat{\delta}(q_1, x) = q_{x_1} \text{ and } \hat{\delta}(q_2, x) = q_{x_2}, \text{ } q_{x_1} \neq q_{x_2}.$$

Since $M$ is trimmed, neither $q_{x_1}$ nor $q_{x_2}$ is a sink state. Thus, there exist strings $v_1, v_2$ such that

$$\hat{\delta}(q_{x_1}, v_1) \in F \text{ and } \hat{\delta}(q_{x_2}, v_2) \in F.$$

Furthermore, since $M$ is minimal, every state in $Q$ is reachable from $q_0$. Hence, there exist strings $w_1, w_2$ such that

$$\hat{\delta}(q_0, w_1) = q_1 \text{ and } \hat{\delta}(q_0, w_2) = q_2.$$

Yet, because $M$ is minimal, $q_{x_1}$ and $q_{x_2}$ must be distinguished by some string. That is, there must exist a string $v_3$ such that $w_1xv_3 \notin L(M)$, but $w_2xv_3 \in L(M)$ or vice versa. Without loss of generality suppose that it is the former.

Then, $w_1xv_1 \in L(M)$ and $w_2xv_3 \in L(M)$, but $w_1xv_3 \notin L(M)$. Hence, $L(M)$ fails to satisfy suffix substitution closure for all $k$.

Thus, by Theorem 1, $L(M)$ is not $SL$, completing the proof of the Lemma.

Next, we prove that if the algorithm assigns a value of $\infty$ to some vertex $v$ in the graph it constructs, then the subgraph rooted at $v$ is cyclic.

**Proof:** If the algorithm assigns a value of $\infty$ to $v$, then there exists a vertex $w$ in the subgraph rooted at $v$, such that a successor of $w$ in the traversal is colored gray.

A vertex is colored gray only if it lies on a path between the root of the subgraph and the vertex the algorithm is currently visiting.

Thus, if $w$ has a successor $x$ that is colored gray, then there is a path (of tree edges) from $x$ to $w$ and an edge from $w$ to $x$, i.e., there is a cycle in the subgraph rooted at $v$.

Now we prove that if the algorithm assigns a value $h \in \mathbb{N}$ to $v$, then the height of the subgraph rooted at $v$ is $h$.

**Proof:** This can be shown by induction.

First, note that if the algorithm assigns a value $h \in \mathbb{N}$ to $v$, then

$$h = \max(\{0\} \cup \{ \text{height}(w) + 1 \mid w \text{ is a successor of } v \}).$$

Thus, if the algorithm assigns 0 to $v$, then $0 \geq \text{height}(w) + 1$ for all $w$ such that $w$ is a successor of $v$. As there can be no such $w$, the height of the subgraph rooted at $v$ is 0.
Now suppose that for all \( h < k \), if the algorithm assigns \( h \in \mathbb{N} \) to \( w \), then the subgraph rooted at \( w \) has height \( h \).

If the algorithm assigns \( k \) to some vertex \( v \), where \( k \neq 0 \), then \( k = h + 1 \) where \( h \) is the height of the tallest subgraph rooted at a successor of \( v \). Hence, \( k \) is the height of the subgraph rooted at \( v \).

Finally, we prove that if the digraph is \( h \in \mathbb{N} \), then \( L(M) \) is SL-\( h+2 \).

Suppose \( h \) is the height of the non-empty digraph and \( x \in \Sigma^* \), \( |x| = h + 1 \). Then \( x = w\sigma \) for some \( w \). Let \( S \) be the vertex at the end of the path labelled \( w \) in the subgraph. Because \( |w| = h \), any successors of \( S \) in the full graph must be singleton. That is,

\[
\text{card}\{\{p \mid \delta(q, \sigma), q \in S\}\} \leq 1.
\]

Thus, there is either no path labeled \( w\sigma \) in the transition graph or all paths \( w\sigma \) lead to the same state. By Lemma 1, \( L(M) \) is SL-\( h+2 \).

### 3.3 Termination

By the definition of a depth first graph search [2], our algorithm will visit each vertex in \( \mathcal{V} \) exactly once. Since \( \mathcal{V} \subseteq \{S \subseteq Q \mid \text{card}(S) \geq 2\} \) and \( \mathcal{V} \) is finite our algorithm can visit only finitely many states and must terminate.

Unfortunately, the worst case for \( \text{card}(\mathcal{V}) \) is \( \Theta(2^{\text{card}(Q)}) \) (in other words, \( \Theta(\text{card}(\mathcal{P}(Q))) \)). Thus, this algorithm’s runtime grows exponentially with \( \text{card}(Q) \).

### 3.4 A Polynomial Time Algorithm

The basic algorithm can be improved, however, since the constructed graph has a great deal of information that is not required to determine whether the stringset is SL. To show that the stringset accepted by an FSA is SL, it is sufficient to show that there exists some \( k \) such that, for all pairs of states \( q_1, q_2 \in Q \) and all paths labelled \( w \) where \( w \in \Sigma^* \) and \( |w| = k - 1 \), there is some state \( q_3 \in Q \) such that \( \delta(q_1, w) = \delta(q_2, w) = q_3 \). This is simply a restatement of Lemma 1.

Given this, it is only necessary to examine all paths among statesets of two FSA states to determine if the stringset accepted by an FSA is SL. The polynomial-time algorithm is nearly identical to the basic algorithm, except that it searches the graph generated by doubleton sets of FSA states (the doubleton graph).

Since the algorithm follows the outedges of each state in the transition graph of the FSA only once for each occurrence of that state in the vertices of the graph traversed in the search, of which there are \( \Theta(\text{card}(Q)^2) \), the time complexity of the doubleton algorithm is \( \Theta(\text{card}(\Sigma) \text{card}(Q)^2) \).

All that remains is to verify that this doubleton graph has the same height as our original graph.

**Lemma 2** If the stringset accepted by the FSA is SL, then the height of the doubleton graph is equal to the height of the graph constructed by the basic algorithm. If the stringset is not SL, then the height of both is \( \infty \).

**Proof:** To show that the height of the graph produced by the basic algorithm is at least as great as the height of the doubleton graph, let \( h \) be the height of the doubleton graph and let \( \{q_1, q_2\}, \{q_3, q_4\} \) be two vertices such that there is a path labelled \( w \) originating at \( \{q_1, q_2\} \) and terminating at \( \{q_3, q_4\} \), where \( |w| = h \).

It can be shown by induction on the length of the path that there exist two vertices of the basic graph, \( R \) and \( S \), such that \( \{q_1, q_2\} \subseteq R \) and \( \{q_3, q_4\} \subseteq S \) and there is a path labelled \( w \) between \( R \) and \( S \).

Thus, when the height of the doubleton graph is \( h \), there is a path of length \( h \) in the basic graph, from which it follows that the height
Figure 3: The Polynomial-time Algorithm

of the basic graph is no less than \( h \).

To show that the height of the doubleton graph is at least as great as that of the graph produced by the basic algorithm, let \( h \) be the height of the basic graph and let \( R, S \) be two vertices such that there is a path labeled \( w \) between \( R \) and \( S \), and \( |w| = h \).

By a similar induction (although working bottom-up) it can be shown that there exist two vertices of the doubleton graph, \( \{q_1, q_2\} \) and \( \{q_3, q_4\} \), such that \( \{q_1, q_2\} \subseteq R \) and \( \{q_3, q_4\} \subseteq S \) and there exists a path labeled \( w \) between \( \{q_1, q_2\} \) and \( \{q_3, q_4\} \).

Since neither graph has greater height than the other, the height of the two graphs must be equal. Thus, if the stringset accepted by the FSA is SL, then the height of the doubleton graph is equal to the height of the graph constructed by the basic algorithm.

Finally, if either graph contains a cycle, a similar construction, based upon the length of the cycle, will show that the other must as well.

It follows from Lemma 2 that the value of \( k \) can be derived from the polynomial-time algorithm just as it can for the original algorithm, that is \( k = h + 2 \) where \( h \) is the height of the forest.

4 Results

We implemented our algorithm and ran it against each phonological pattern in Heinz’s catalog. Of the 109 patterns:

- 9 are SL\(_2\)
- 44 are SL\(_3\)
- 24 are SL\(_4\)
- 3 are SL\(_5\)
- 1 is SL\(_6\)
- 28 are not SL.

Three quarters of the phonological patterns turn out to be simple, many requiring a scanning window of only size 3 to be recognized. The highest \( k \) value we found, a \( k \) of 6, fits well within common assumptions about working memory.
5 Conclusion and Future Work

We have given our simplified algorithm, implemented it, and applied it to Heinz’s catalog. The majority of the patterns turn out to be SL with a $k$ bounded by 6. Thus a cognitive mechanism which recognize those patterns needs only to be sensitive to the last 6 symbols or sounds it encounters. And to learn those patterns, such a mechanism needs only to memorize those $k$ factors, meaning that while the brain does not simply memorize words to learn lexicons, it could do so simply by memorizing strings of $k$ symbols instead of entire words.

Though it may seem that the non-SL patterns need an arbitrarily longer scanning window, this is not the case; instead a more sophisticated recognition strategy is needed. Our current research is testing these remaining non-SL patterns against the next level of the Sub-Regular Hierarchy, the Locally Testable (LT) stringsets.

References


