# An algebraic characterization of strictly piecewise languages*

Jie Fu, Jeffrey Heinz, and Herbert G. Tanner

University of Delaware
jiefu,heinz,btanner@udel.edu

**Abstract.** This paper provides an algebraic characterization of the Strictly Piecewise class of languages studied by Rogers et al. 2010. These language are a natural subclass of the Piecewise Testable languages (Simon 1975) and are relevant to natural language. The algebraic characterization highlights a similarity between the Strictly Piecewise and Strictly Local languages, and also leads to a procedure which can decide whether a regular language $L$ is Strictly Piecewise in polynomial time in the size of the syntactic monoid for $L$.

## 1 Introduction

Rogers et al. [12] study the Strictly Piecewise (SP), which are a proper subclass of the Piecewise Testable (PT) languages of Simon [13]. The Strictly Piecewise languages are interesting for two reasons. First, there are several senses in which the SP class is natural. For example, SP is exactly the class of those languages closed under subsequence [12]. Also, they bear the same relation to Piecewise Testable languages that the Strictly Local (SL) bear to Locally Testable (LT) languages [10,12]. Second, this class expresses some of the kinds of long-distance dependencies found in natural language [6, 12].

While Rogers et al. provide several characterizations of SP languages, they do not provide an algebraic one. Also, the procedure they give for deciding whether a regular language $L$ belongs to SP is exponential in the size of the smallest deterministic acceptor for $L$. This paper addresses these issues. It provides an algebraic characterization for the SP class. This result not only reveals an important similarity between the SP and SL languages, but also leads to a procedure which decides whether $L$ belongs to SP in time quadratic in the size of syntactic monoid for $L$. However, it remains an open question whether a polynomial time decision procedure exists in the size of the smallest deterministic acceptor.

The rest of this paper is organized as follows. Section 2 reviews foundational concepts and notation. Section 3 defines the Piecewise Testable (PT), Strictly Piecewise (SP), and Stricly Local (SL) classes. Section 4 presents our algebraic characterization of the SP class and Section 5 describes the polynomial-time decision procedure. Finally, Section 6 concludes.

* This research is supported by grant #1035577 from the National Science Foundation.

## 2 Preliminaries

A *semigroup* is a set with an associative operation. A *monoid* is a semigroup with an identity element (written 1). If $S$ is a semigroup, $S^1$ denotes the monoid equal to $S$ if $1 \in S$ and to $S \cup \{1\}$ otherwise. A *zero* is an element 0 such that, for every $s \in S$, $s0 = 0s = 0$. The *free* semigroup (monoid) of a set $S$ is the set of all finite sequences of one (zero) or more elements from $S$.

If $x$ is an element of set $S$ and $\pi$ a partition of $S$, the *block* of $\pi$ containing $x$ is $[x]_\pi$. The partition of $S$ induced by an equivalence relation $\rho$ is $S/\rho$. A *right (left) congruence* is a partition such that if $[x]_\pi = [y]_\pi$ then $[xz]_\pi = [yz]_\pi$ ($[zx]_\pi = [zy]_\pi$). A *congruence* is both a left and a right congruence.

Following Clifford [2], a *left (right) ideal* of a semigroup $S$ is a non-empty subset $T$ of $S$ such that $ST \subseteq T$ ($TS \subseteq T$). The *left (right) ideal* of $S$ *generated by* $T$ is $T \cup ST = S^1T$ ($T \cup TS = TS^1$). The *principal left (right) ideal* of $S$ generated by $t \in T$ is $PL(t) = S^1t$ ($PR(t) = tS^1$).

Let $\Sigma$ denote a finite set, called the *alphabet*. Sets $\Sigma^+$ and $\Sigma^*$ denote the free semigroup and free monoid of $\Sigma$, respectively. We refer to the elements of $\Sigma^+$ and $\Sigma^*$ as *strings* and *words* interchangeably. The unique string of length zero is denoted $\lambda$. The set $\Sigma^{\leq k}$ denotes the set of all words of length at most $k$.

The length of a string $u$ is denoted $|u|$, and $|w|_\sigma$ denotes the number of occurences of $\sigma$ in $w$. A string $v$ is a *factor* of $w$ iff there exist strings $x, y \in \Sigma^*$ such that $w = xvy$. A string $v$ is a *prefix* (*suffix*) of $w$ iff there exist $x \in \Sigma^*$ such that $w = vx$ ($w = xv$). A string $v$ is a *subsequence* of string $w$ iff $v = \sigma_1 \cdots \sigma_n$ and $w \in \Sigma^*\sigma_1\Sigma^* \cdots \Sigma^*\sigma_n\Sigma^*$, and we write $v \sqsubseteq w$. Languages are subsets of $\Sigma^*$. The complement of a language $L$ is $\overline{L} = \{w \in \Sigma^* : w \notin L\}$.

A *semiautomaton* is a tuple $A = \{Q, \Sigma, T\}$, where $Q$ is a non-empty finite set of states and $\Sigma$ is the alphabet. The transition function is a partial function $T : Q \times \Sigma \to Q$. The domain of the transition function is expanded to $Q \times \Sigma^*$ recursively as follows. For all $q \in Q$, $T(q, \lambda) = q$ and for all $w \in \Sigma^*$ and $\sigma \in \Sigma$, $T(q, wa) = T(T(q, w), a)$. It follows that $T(q, xy) = T(T(q, x), y)$. By definition semi-automata are deterministic.

A finite-state automaton (FSA) is a tuple $\mathbf{A} = \{Q, q_0, Q_f, \Sigma, T\}$, where $\{Q, \Sigma, T\}$ is a semi-automaton, $q_0 \in Q$ is the initial state, and $Q_f \subseteq Q$ is a set of final states. The language recognized by $\mathbf{A}$ is $\{w \in \Sigma^* : T(q_0, w) \in Q_f\}$.

A language $L$ is *regular* iff there exists a FSA recognizing it. For every regular language $L$ there is a unique (up to isomorphism) automaton with the fewest number of states recognizing $L$ called the *canonical* FSA for $L$.

A state $q$ of an automaton is a *sink* state iff, for all $\sigma \in \Sigma$, if $T(q, \sigma)$ is defined then $T(q, \sigma) = q$. One can always make the transition function total by adding a nonfinal sink state and directing all the missing transitions for each state to this sink. When the sink state is added to a canonical acceptor, it is the only state which is both a sink and nonfinal. The resulting automaton is *complete*.

For any automaton $\mathbf{A}$ and state $q \in Q$, let $\rho_q$ be that relation such that, for all elements $x$ and $y$ of $\Sigma^*$, $x \rho_q y$ iff $T(q,x) = T(q,y)$. More generally, let

$$f_x = \begin{pmatrix} q_1 & \cdots & q_n \\ T(q_1,x) & \cdots & T(q_n,x) \end{pmatrix} .$$

For all $x, y \in \Sigma^*$, let $x \rho y$ iff $f_x = f_y$. The equivalence relation $\rho$ over $\Sigma^*$ induces a congruence over $\Sigma^*$ [15]. The index of $\rho$ is finite because $Q$ is finite.

Let $F_A = \{f_x : x \in \Sigma^*\}$ denote the finite monoid of mappings and $\bar{I}(A) = \Sigma^*/\rho$. Then $F_A$ is isomorphic to $\bar{I}(A)$ under the correspondence of $f_x$ of $F_A$ with $[x]$ of $\bar{I}(A)$, where $[x]$ is the $\rho$-congruence coset containing $x$ of $\Sigma^*$. In this paper, when writing $f_x$ and $[x]$, we choose $x$ to be a shortest-length element in the congruence class without any ambiguity.

For FSA $\mathbf{A}$, where $A$ is the associated semiautomaton of $\mathbf{A}$, $F_A$ is called the *transformation semigroup* and $\bar{I}(A)$ is the *characteristic semigroup* of $A$. Elements $f_x$ of $F_A$ can also be written in matrix form $\mu_x$, where the rows and columns indicate states in $Q = \{q_1, \ldots, q_n\}$ and $\mu_x[i,j] = 1$ iff $T(q_i, x) = q_j$.

The set of matrices is another semigroup, the *transition semigroup*. The name is derived from the fact that each element in this semigroup is a transition matrix associated to a *walk* $x$ in $\mathbf{A}$. We write $U_A = \{\mu_x : x \in \Sigma^*\}$. Clearly $U_A$ is isomorphic to $\bar{I}(A)$ under the correspondance of $\mu_x$ of $U_A$ with $[x]$ of $\bar{I}(A)$.

**Definition 1 (Pin 1997).** *The* syntactic semigroup *of a regular language $L$ is the transformation semigroup given by its complete canonical semiautomaton.*

In the syntactic semigroup of an automaton $A$, the *set of generators* of $F_A$ is $Gen(F_A) = \{f_\sigma : \sigma \in \Sigma\}$. The *syntactic monoid* of a regular language $L$ is the syntactic semigroup with identity, $Gen(F_A^1) = \{f_\sigma : \sigma \in \Sigma \cup \lambda\}$.

Pin [11] discusses the equivalence between automata and semigroups. Note that since the transition semigroup $U_A$ of $\mathbf{A}$ is represented as a semigroup of boolean matrices of order $|Q| \times |Q|$, a word $w$ is recognized by $\mathbf{A}$ iff $\mu_x(q_0, q_f) = 1$ for some final state $q_f \in Q_f$. It follows that a finite automaton recognizes a regular language $L$ iff its transition semigroup recognizes $L$.

A "monoid graph" is a useful method employed by contemporary algebraic theorists to visualize monoids. The nodes of the graph are elements in the monoid, though an initial node labeled "$\lambda$" is included by convention. The labels on edges are the elements in the set of generators of the monoid. Given a monoid $M$, $x \xrightarrow{s} y$ iff $xs = y$, where $x, y \in M$, and $s \in Gen(M)$. The monoid graph of $F_A$ is denoted as $\mathrm{MG}(F_A)$. We mark elements $x$ in the monoid graph as final iff $f_x \in F_A$ and there exists a final state $q$ in the canonical acceptor such that $T(q_0, x) = q$ [11]. Examples of monoid graphs are in Figures 1,2, and 3.

**Definition 2.** *A unique nonfinal sink state in an automaton $\boldsymbol{A}$ is called* zero. *An element $f_x$ is a zero element of the transformation semigroup iff*

$$f_x = \begin{pmatrix} q_1 \ldots q_n \\ \boldsymbol{0} \ldots \boldsymbol{0} \end{pmatrix} .$$

*We use the notation $f_x = 0$ for the transformation semigroup, $\mu_x = 0$ for the transition semigroup, and $x = 0$ for the free semigroup $\Sigma^*$. The corresponding zero in the characteristic semigroup $\bar{I}(A)$ is denoted* [0].

While every complete canonical automaton (except the one recognizing $\Sigma^*$) has a unique nonfinal sink state, not every transformation semigroup has a zero.

## 3 Piecewise Testable and Strictly Piecewise languages

The concept of a subsequence is central to the notion of piecewise testability.

**Definition 3.** *The* principle shuffle ideal *of $v$ is the language of all words for which $v$ is a subsequence. We write $SI(v) = \{w \in \Sigma^* \mid v \sqsubseteq w\}$.*

The Piecewise Testable languages is the smallest class of languages including SI($w$) for all $w \in \Sigma^*$ and closed under Boolean operations [13]. Similarly, the class of Piecewise $k$-Testable (PT$_k$) languages is the smallest class of languages including $SI(w)$ for all $w \in \Sigma^{\leq k}$ and closed under Boolean operations.

A well-known characterization of the PT languages is stated in terms of the sets of subsequences within words. If $P_{\leq k}(w) \stackrel{\text{def}}{=} \{v : v \sqsubseteq w \text{ and } |v| \leq k\}$ then the following characterization (sometimes taken as the definition of PT [3]) holds.

**Theorem 1.** *A language $L$ is Piecewise Testable iff there exists $k$ such that, for all words $w_1, w_2 \in \Sigma^*$, if $P_{\leq k}(w_1) = P_{\leq k}(w_2)$ then $w_1 \in L$ iff $w_2 \in L$.*

When $k$ is known, $L$ is said to be Piecewise $k$-Testable ($L \in PT_k$).

Simon proved one of the first examples of what later became known as Eilenberg's correspondence theorem [11]. One of the relations that Green [4] defines on semigroups is the $\mathcal{J}$ relation, which relates two elements of a semigroup $S$ if they generate the same two-sided principal ideal of $S$: $a\mathcal{J}b$ iff $S^1aS^1 = S^1bS^1$. A semigroup $S$ is *$\mathcal{J}$-trivial* iff, for all $a, b \in S$, if $a\mathcal{J}b$ then $a = b$. Simon proved the following algebraic characterization of piecewise testable languages.

**Theorem 2 (Simon 1975).** *A language is Piecewise Testable iff its syntactic monoid is $\mathcal{J}$-trivial.*

As an example, consider the language of all words with exactly one $a$, $L = \{w : |w|_a = 1\}$. The canonical acceptor for this language is shown in Figure 1. There are three elements in the monoid $F^1_{A_1} = \{a, 1, 0\}$, (for simplicity of notation, let $x$ stand for $f_x$). The $\mathcal{J}$-triviality is established by calculating $F^1_{A_1} x F^1_{A_1}$, for all $x \in F^1_{A_1}$: $F^1_{A_1} a F^1_{A_1} = \{0, a\}$, $F^1_{A_1} 1 F^1_{A_1} = \{0, a, 1\}$, and $F^1_{A_1} 0 F^1_{A_1} = \{0\}$. The $\mathcal{J}$-triviality is satisfied, which means this languange $L$ is piecewise testable.

Rogers et al. [12] study a proper subclass of the Piecewise Testable languages, the *Strictly Piecewise* class. This paper takes as definition of Strictly Piecewise languages those languages which are closed under subsequence. (Unknown to Rogers et al., languages closed under subsequence were studied forty years earlier by Haines [5] (see also Higman [7]).)

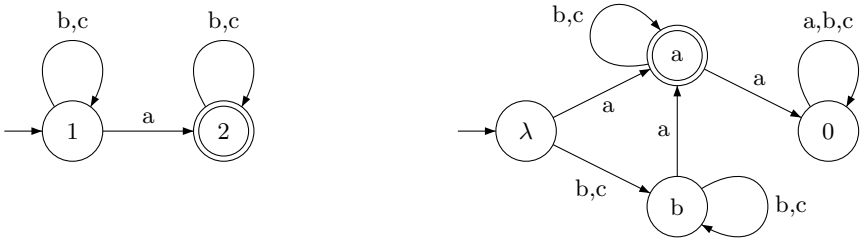


**Fig. 1.** The canonical automaton and the monoid graph for $L = \{w : |w|_a = 1\}$, which is the language of all words with exactly one $a$.

**Definition 4.** *A language is Piecewise Testable in the Strict Sense ($L \in \mathrm{SP}$) iff, for all $w \in \Sigma^*$, if $w \in L$ and $v \sqsubseteq w$ then $v \in L$.*

Rogers et al. [12] establish the following equivalences (see also [5]).

**Theorem 3.** *The following are equivalent:*

1. $L \in \mathrm{SP}$.
2. $L = \overline{\mathrm{SI}(X)}, X \subseteq \Sigma^*$.
3. $L \in \bigcap_{w \in S} \overline{\mathrm{SI}(w)}$, *for $S$ finite.*
4. *there exists $k$ such that if $P_{\leq k}(w) \subseteq P_{\leq k}(L)$ then $w \in L$.*

It follows from the third characterization above that any SP language can be characterized by a finite set $S$. Elements of this set are the *forbidden* subsequences, and the language is all words which do not contain any of these *forbidden* subsequences. The longest word in $S$ is the length $k$ in the 4th characterization above, in which case we say $L$ is Strictly $k$-Piecewise ($L \in SP_k$).[1]

By forbidding subsequences, SP languages resemble the Stricly Local languages which forbid factors [10]. Any SL language $L$ can be defined as the intersection of the complements of sets defined to be those words which *contain* a forbidden factor. Formally, let the *container* of $w \in \rtimes\Sigma^*\ltimes$ be $C(w) = \{u \in \Sigma^* : w \text{ is a factor of } \rtimes u \ltimes\}$ then a language $L \in SL$ iff there exists a finite set of forbidden factors $S \subset \rtimes\Sigma^*\ltimes$ such that $L = \bigcap_{w \in S} \overline{C(w)}$.[2] Figure 2 shows the canonical acceptor and the monoid graph for the SL language $L = \overline{\Sigma^* aa \Sigma^*} = \overline{C(aa)}$, i.e. all words except those containing the factor $aa$.

To illustrate SP languages, consider the language $L = \overline{\mathrm{SI}(bb)} \cap \overline{\mathrm{SI}(ca)}$, which is the language of all words except those containing either the subsequences $bb$ or $ca$; i.e., $bb$ and $ca$ are the forbidden subsequences. Thus this SP language can be characterized by the set $\{bb, ca\}$ of forbidden subsequences (or equivalently by the set $\Sigma^{\leq 2}/\{bb, ca\}$ of permissible subsequences [12]). Hence this language belongs to $\mathrm{SP}_2$. Figure 3 shows ths canonical automata and the monoid graph for $L$. The 0 element is not shown there, but note that all missing edges go to 0.

[1] While every SP language is convex [14], it is not the case that all convex languages are SP since, for example, there are nonempty subword-convex languages that do not contain $\lambda$ but the only SP language not containing $\lambda$ is the empty one.

[2] The symbols $\rtimes$ and $\ltimes$ invoke left and right word boundaries and are necessary because SL languages make distinctions at word edges [10].

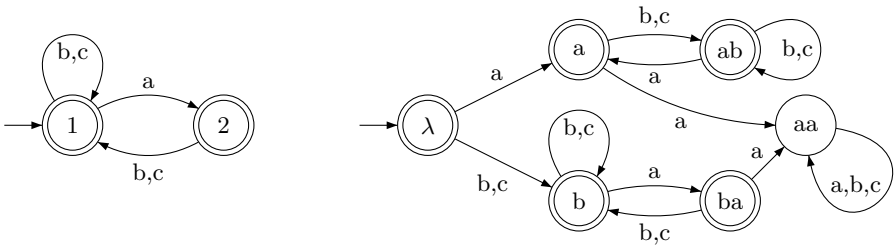


**Fig. 2.** The canonical acceptor and the monoid for the language $L = \overline{C(aa)}$, which is all words except those containing the factor $aa$.

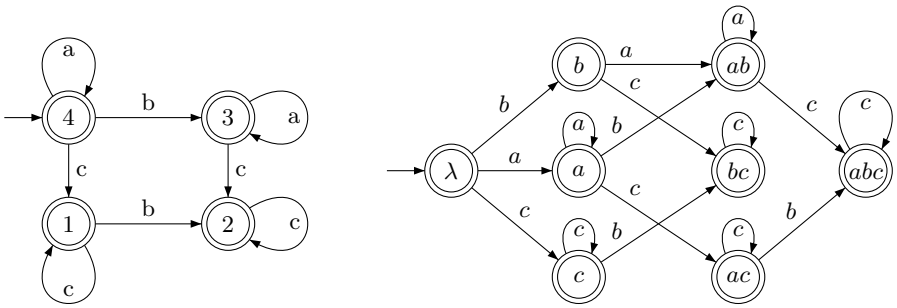


**Fig. 3.** The canonical automata and the monoid graph of the syntactic monoid of $L = \overline{\mathrm{SI}(bb)} \cap \overline{\mathrm{SI}(ca)}$, i.e. the language where the subsequences $bb$ and $ca$ are forbidden.

As with the other piecewise testable languages like the one in Figure 1, it is not difficult to verify that the syntactic monoid of this language is $\mathcal{J}$-trivial. However, this language, like every other SP language, has two additional properties. Furthermore, no non-SP language has both of these properties.

## 4 Algebraic Characterization of SP

There are two important concepts that need to be introduced.

**Definition 5.** *Let $L$ be a regular language recognized by FSA, and consider its characteristic semigroup. Language $L$ is* wholly nonzero *if and only if $\overline{L} = [0]$.*

In other words, a language is wholly nonzero if and only if every word not in the language is in the zero block of the characteristic semigroup. In terms of the transformation semigroup, this means that every word $x$ not in the language is zero; i.e., $f_x = 0$.

**Theorem 4.** *A language $L$ is wholly nonzero if and only if $L$ is closed under prefix and closed under suffix.*

*Proof.* Clearly, $[0] \subseteq \overline{L}$. Now suppose $L$ is closed under prefix and suffix, and consider any $x \in \overline{L}$. For contradiction, suppose $f_x \neq 0$. Then in the canonical acceptor $\mathbf{A}$ for $L$ there are states $q, q'$ in $\mathbf{A}$ such that $x$ transforms $q$ to $q'$. Since

**A** is canonical, there exist strings $w, y$ such that $w$ transforms $q_0$ to $q$ and $y$ transforms $q'$ to a final state. Thus $wxy \in L$. Since $L$ is closed under prefix $wx$ belongs to $L$ and since $L$ is closed under suffix, $x$ belongs to $L$, which contradicts the assumption. Therefore $f_x = 0$, which completes one direction of the proof.

Now suppose $\overline{L} = [0]$ and consider any $w \in L$ and any prefix (suffix) $v$ of $w$, which means there exists $x$ such that $w = vx$ ($w = xv$). If $v \notin L$ then by assumption $f_v = 0$. It follows that $f_w = f_{vx} = 0f_x = 0$ ($f_w = f_{xv} = f_x 0 = 0$), which contradicts that $w \in L$.

Observe that $L = \Sigma^*$ and the empty language are wholly nonzero vacuously.

The following two corollaries are almost immediate.

**Corollary 1.** *The Strictly Piecewise languages are wholly nonzero.*

*Proof.* The Strictly Piecewise are closed under subsequence by definition and are therefore closed under prefix and suffix.

**Corollary 2.** *The Strictly Local languages are wholly nonzero.*

*Proof.* Consider any Strictly Local language $L$ and any $w \in L$. Since $w \in L$, there are no forbidden factors in $w$ and therefore there are none in any prefix or suffix of $w$. Hence every prefix and suffix of $w$ belongs to $L$ as well.

That both the Strictly Local and Strictly Piecewise are wholly nonzero is a nontrivial property they have in common. To illustrate, recall the SL language $L = \overline{C(aa)}$ (Figure 2). Every string not in this SL language transforms any state in its monoid graph to 0. These are all the strings with the 2-factor $aa$. Similarly, consider again the language $L = \overline{\mathrm{SI}(bb)} \cap \overline{\mathrm{SI}(ca)}$ (Figure 3). Every string not in this SP language transforms any state in its monoid graph to 0. These are exactly those strings with either subsequence $bb$ or $ca$.

The second property is an algebraic characterization of what Rogers et al. describe in automata-theoretic terms as "missing edges propagate down." This means that if some state $q$ in the canonical accepter does not have a transition labeled with symbol $\sigma$ then no state reachable from $q$ has an outgoing transition labeled with $\sigma$. To capture this, we need the following concept relating to zeroes.

**Definition 6.** *Let $M$ be a monoid. The set of* right annihilators *of an element $x \in M$, is $RA(x) = \{a \in M : xa = 0\}$.*

In other words, the elements of $RA(x)$ *annihilate* $x$ from the right. The set of *left annihilators* can be defined similarly, but it does not play a role here.

We now define the following property which captures the notion of "missing edges propagating down."

**Definition 7.** *A language $L$ is* right annihilating *iff for any element $f_x$ in the syntactic monoid $F_A(L)$, and for all $f_w$ in the principle right ideal generated by $f_x$, it is the case that $RA_{F_A(L)}(f_x) \subseteq RA_{F_A(L)}(f_w)$.*

The main theorem of this paper can now be stated and proved.

**Theorem 5.** *A language $L$ is* SP *iff $L$ is wholly nonzero and right annihilating.*

*Proof.* By Corollary 1, any SP language is wholly nonzero.

Next consider any $L \in$ SP and any element $f_x$ and any $f_t \in RA_{F_A(L)}(f_x)$. It follows that $f_x f_t = 0$; hence, $f_{xt} = 0$. Since $L \in$ SP, there must be some $v \sqsubseteq xt$ such that $v$ is forbidden; i.e SI$(v) \subseteq \overline{L}$. For any $f_w$ in the principal right ideal of $f_x$, it is the case that there exists $f_a$ such that $f_w = f_x f_a$. Thus $f_w f_t = f_x f_a f_t = f_{xat}$. Since $v \sqsubseteq xt$ it follows that $v \sqsubseteq xat$ and therefore $f_w f_t = f_{xat} = 0$ and so $f_t \in RA_{F_A(L)}(f_w)$. The generality of $f_w$ and $f_t$ ensures that $\forall w \in PR(x), RA_{F_A(L)}(f_x) \subseteq RA_{F_A(L)}(f_w)$.

Now for the other direction. The empty language vacuously satisfies the above conditions and belongs to SP so consider any nonempty regular language $L$, which is wholly nonzero and right annihilating. We show that $L$ belongs to SP.

By contradiction, suppose $L$ is wholly nonzero and right annihilating, but not in SP. By definition of SP, L is not closed under subsequence. So there is some $w$ and $v$ such that $w \in L$ and $v \sqsubseteq w$ but $v \notin L$. Since $v \sqsubseteq w$, there exists $u_0, u_1, \cdots, u_n$ such that for $v = \sigma_1\sigma_2 \ldots \sigma_n$, $w = u_0\sigma_1 u_1\sigma_2 u_2 \cdots \sigma_n u_n$.

Since $v \notin L$ and since $L$ is wholly nonzero, $v \in [0]$. It will be useful to refer to the suffixes of $v$ as follows: $v_i = \sigma_i \cdots \sigma_n$ for $1 \leq i \leq n$. For example, $v = v_1 = \sigma_1 \cdots \sigma_n$ and $v_2 = \sigma_2 \cdots \sigma_n$, and $v_n = \sigma_n$.

Now $v_2$ is a right annihilator of $u_0\sigma_1$ since $u_0\sigma_1 v_2 = u_0 v = u_0 0 = 0$. Also, since $L$ is right annihilating, $RA(u_0\sigma_1)$ is a subset of $RA(u_0\sigma_1 u_1)$, and so $v_2$ right annihilates $u_0\sigma_1 u_1$ as well.

Next consider that $v_3$ is a right annihilator of $u_0\sigma_1 u_1\sigma_2$ since $u_0\sigma_1 u_1\sigma_2 v_3 = u_0\sigma_1 u_1 v_2$ and above we showed that $v_2$ right annihilates $u_0\sigma_1 u_1$. Again, since $L$ is right annihilating, $RA(u_0\sigma_1 u_2\sigma_2)$ is a subset of $RA(u_0\sigma_1 u_1\sigma_2 u_2)$, and so $v_3$ right annihilates $u_0\sigma_1 u_1\sigma_2 u_2$ as well.

Carrying this argument through to its conclusion, we see that $v_n = \sigma_n$ is a right annihilator of $u_0\sigma_1 u_1\sigma_2 u_2 \cdots u_{n-1}\sigma_{n-1}$. Therefore $\sigma_n$ is a right annihilator of $u_0\sigma_1 u_1\sigma_2 u_2 \cdots u_{n-2}\sigma_{n-1}u_n$ as well. Hence $u_0\sigma_1 u_1\sigma_2 u_2 \cdots u_{n-2}\sigma_{n-1}u_n\sigma_n = 0$.

But this means that $w = u_0\sigma_1 u_1\sigma_2 u_2 \cdots u_{n-2}\sigma_{n-1}u_n\sigma_n u_n = 0u_n = 0$. Since $L$ is wholly nonzero, it follows that $w \notin L$, which contradicts the reductio assumption. Therefore there is no $v, w$ such that $w \in L$, $v \sqsubseteq w$, and $v \notin L$. It follows that regular languages that are wholly nonzero and right annihilating are closed under subsequence and are therefore SP.

We illustrate this property in the context of the decision procedure we present below for deciding whether a regular language is SP.

## 5 Algorithms for SP languages

Theorem 3 provides a polynomial-time decision procedure for deciding whether any regular language $L$ is SP, and if it is, it finds the finite set of the shortest forbidden subsequences necessary to define $L$.

### 5.1 Deciding SP

The input to the algorithms below is taken to be the monoid graph of the syntactic monoid for a regular language $L$, with the initial state being the node labeled "$\lambda$" and the final states being marked. Since this graph is determinstic, it is possible to obtain the canonical acceptor in time $O(n \log n)$ [9]. Given a minimal DFA $\mathbf{A}$, the syntactic monoid $F_A$ can be obtained through the set of generators $\{f_\sigma\}, \forall \sigma \in \Sigma$. The reader is referred to [1] for the construction method of syntactic monoid $F_A$.

Theorem 3 provides the basis for the decision procedure, which we call DSP. DSP simply checks whether the syntactic monoid satisfies the wholly nonzero and the right annihilation conditions.

The wholly nonzero condition can be checked in two steps, essentially by checking closure under prefixes and suffixes. To check closure under prefixes, one simply need check whether every state in the canonical accepter $\mathbf{A}$ is final. If they are not, then the syntactic monoid of $\mathbf{A}$ is not wholly nonzero. To check closure under suffixes, both the complete canonical acceptor and the transformation semigroup $F_A$ are examined. Let 0 be the non-final sink state in the complete automaton. If there exists one nonzero element $f_x$ in $F_A$ and one noninitial state $q$ in the canonical acceptor such that $T(q, x) \neq 0$ but $T(q_0, x) = 0$, then the wholly nonzero condition is violated. If no such $f_x$ or $q$ exist, however, we can conclude the language is wholly nonzero.

Whether the right annihilating condition is satisfied can be determined from the Cayley table for $F_A$. The columns and rows of a Cayley table are labeled with the elements in the syntactic monoid $F_A$, and the cell is the product$(x \cdot y)$ of the row-th$(x)$ and column-th$(y)$ elements [2]. Then for each $f_x \in F_A$, the principal right ideal generated by $x$ ($PR(x)$) can be found by the union of all distinct elements in the $x$th row of the table and the right annhilators of $x$ ($RA(x)$) are given by those elements $y$ such that the $x^{\text{th}}$ row and $y^{\text{th}}$ column is 0.

Then for each $z \in PR(x)$, it is sufficient to check whether $RA(x) \subseteq RA(z)$. If for any $x \in F_A$ and any $z \in PR(x)$, it is the case that $RA(x) \not\subseteq RA(z)$ then the algorithm exits and returns "false". Otherwise it returns "true".

We illustrate these procedures with three examples. Consider first the SP language $L = \overline{\text{SI}(bb)} \cap \overline{\text{SI}(ca)}$ in Figure 3. The elements of its transformation semigroup $F_A(L) = \{f_x, x \in \Sigma^*\}$ are:

$$f_a = \begin{pmatrix} 1\,2\,3\,4 \\ 0\,0\,3\,4 \end{pmatrix} \qquad f_b = \begin{pmatrix} 1\,2\,3\,4 \\ 2\,0\,0\,3 \end{pmatrix} \qquad f_c = \begin{pmatrix} 1\,2\,3\,4 \\ 1\,2\,2\,1 \end{pmatrix}$$

$$f_{ab} = \begin{pmatrix} 1\,2\,3\,4 \\ 0\,0\,0\,3 \end{pmatrix} \qquad f_{bc} = \begin{pmatrix} 1\,2\,3\,4 \\ 2\,0\,0\,2 \end{pmatrix} \qquad f_{ac} = \begin{pmatrix} 1\,2\,3\,4 \\ 0\,0\,2\,1 \end{pmatrix}$$

$$f_{abc} = \begin{pmatrix} 1\,2\,3\,4 \\ 0\,0\,0\,2 \end{pmatrix} \qquad 0 = \begin{pmatrix} 1\,2\,3\,4 \\ 0\,0\,0\,0 \end{pmatrix} .$$

Since $F_A$ is isomorphic to the characteristic semigroup $\bar{I}(A)$, it follows that $\bar{I}(A) = \{[0], [a], [b], [c], [ab], [bc], [ac], [abc]\}$. The transition semigroup $U(A)$ are the set of the adjacency matrices given by each string $x$ in $f_x, f_x \in F_A$.

| | $\lambda$ | a | b | c | ab | bc | ac | abc |
|---|---|---|---|---|---|---|---|---|
| $\lambda$ | $\lambda$ | a | b | c | ab | bc | ac | abc |
| a | a | a | ab | ac | ab | abc | ac | abc |
| b | b | ab | 0 | bc | 0 | 0 | abc | 0 |
| c | c | 0 | bc | c | 0 | bc | 0 | 0 |
| ab | ab | ab | 0 | abc | 0 | 0 | abc | 0 |
| bc | bc | 0 | 0 | bc | 0 | 0 | 0 | 0 |
| ac | ac | 0 | abc | ac | 0 | abc | 0 | 0 |
| abc | abc | 0 | 0 | abc | 0 | 0 | 0 | 0 |

**Table 1.** Cayley table for syntactic monoid for $L = \overline{\mathrm{SI}(bb)} \cap \overline{\mathrm{SI}(ca)}$.

The monoid graph for this language is in Figure 3. Recall that although the 0 element is not shown, it is understood that all missing edges go to 0. The Cayley table is given in Table 1. With a little abuse of notation, in the following context, $x$ is used to denote the element $f_x$ in syntactic monoid $F_A$. The wholly non-zero condition can be checked by examining the syntactic monoid. It is noticed that in this canonical accepter all states are finals and there is no such $f_x \in F_A$ and $q \in Q$ such that $T(q, x) \neq 0$ but $T(q_0, x) = 0$.

The next step is to determine whether the right annihilation condition is satisfied with the help of Cayley table. For example in the Cayley table, the $ab$-row is all the elements that are in the right ideal generated by $ab$, $abF_A^1 = \{ab, abc, 0\}$. The elements in those columns corresponding to 0s form the set $RA(ab) = \{b, ab, bc, abc\}$. The right annihilating condition requires that $\forall w \in xF_A$, $RA(x) \subseteq RA(w)$. From the table it is easy to verify that $RA(abc) = \{a, b, ab, bc, ac, abc\}$, which is a superset of $RA(ab)$. Since $RA(0) = F_A$, it likewise follows that $RA(ab) \subseteq RA(0)$. The right annihilation condition for other elements can be verified in the same manner and it can be shown this syntactic monoid is right-annhilating.

Now consider the language $L = \{w : |w|_a = 1\}$ (Figure 1). $L$ is not SP because it does not satisfy the wholly nonzero condition. The element $b$ is not in the language but it is not zero in its syntactic semigroup.

For the language $L = \overline{C(aa)}$ in Figure 2, though it satisfies the wholly nonzero condition, the right annihilating condition is violated. Observe that $aa = 0$ and $ab \in PR(a)$. If $L$ were right annihilating then $RA(a) \subseteq RA(ab)$. However, $aba = a \neq 0$ and thus the right annihilating condition is not met. Therefore, $L = \overline{C(aa)}$ is not SP.

### 5.2 Finding the shortest forbidden subsequences

The following procedure FIND-SSQ takes the syntactic monoid of a SP language as input and finds the finite set of shortest forbidden subsequences which describe the SP language. In order to link the syntactic monoid and the length of forbidden subsequences, the monoid graph is employed. In $\mathrm{MG}(F_A)$, the set of the shortest paths that lead to 0 of $F_A$ form the set $\mathcal{P}(F_A)$.

FIND-SSQ begins with the syntactic monoid for some $L \in \mathrm{SP}$ and $k = 1$.

1. Calculate the set of sets of $k$-subsequences:
$$P_k(\mathcal{P}(F_A)) \stackrel{\text{def}}{=} \left\{\{P_k(p)\} : p \in \mathcal{P}(F_A)\right\}.$$
2. Find all singleton sets in $P_k(\mathcal{P}(F_A))$ and construct the set $FS_k$, which is the set of hypothesized forbidden subsequences of length $k$. This set is formed by taking the union of the singleton sets in $P_k(\mathcal{P}(F_A))$. If there is no singleton set found, update $k$ by one and return to step 1.
3. Verify whether each set $P \in P_k\big(\mathcal{P}(F_A)\big)$ has a nonempty intersection with $FS_k$. If every set $P$ does then $FS_k$ is a set of forbidden sequences which can define $L$ and $L$ is Strictly $k$-Piecewise. Otherwise, update $k$ by one and return to step 1.

**Theorem 6.** FIND-SSQ *terminates at the shortest $k$ for $L \in \mathrm{SP}$.*

*Proof.* Suppose this $k$ is not the shortest one for the SP language $L$, and there exists $k' > k$ such that $L \in \mathrm{SP}_{k'}$. This means that there exists at least one path $p' \in \mathcal{P}(F_A)$, with $|p'| > k$, such that $P_k(p') \subseteq P_k(L)$ and $P_{k'}(p') \cap P_{k'}(L) = \emptyset$, for some $k' > k$. The fact that $P_k(p') \subseteq P_k(L)$ implies that $\forall v \in P_k(p')$, $v \in L$, which is guaranteed by the syntactic monoid of $L$ being wholly nonzero.

However, if the algorithm does not terminate at $k$ ensures that there exists at least one element $h \in P_k(p')$ with $h \in FS_k$. Since $FS_k$ is the set of all paths of length $k$ that lead to 0, $h \notin L$. This contradicts the previous statement that $\forall v \in P_k(p')$, $v \in L$. Therefore, no such $p'$ exists and thus the algorithm terminates at the shortest $k$ for the strictly piecewise language $L$.

We illustrate this algorithm with the automaton in Figure 3, assuming it has already been verified with DSP that it describes an SP language. We refer to the monoid in Figure 3 with $F_A$. The set of the shortest paths which lead to 0 is $\mathcal{P}(F_A) = \{bb, ca, bab, bcb, bca, abb, aca, cba, cbb, bacb, baca, abcb, abca, acbb, acba\}$.

1. For $k = 1$, all sets in $P_{=1}(\mathcal{P}(F_A))$ are not singleton. Therefore, increase $k$ by 1.
2. For $k = 2$, $P_{=2}(\mathcal{P}(F_A)) = \big\{\{bb\}, \{ca\}, \{ba, ab, bb\}, \{bc, cb, bb\}, \{bc, ba, ca\}, \{ab, bb\}, \{ac, aa, ca\}, \{cb, ca, ba\}, \{cb, bb\}, \{ba, bc, ac, bb, ab, cb\}, \{ab, bc, ac, cb, bb\}, \{ab, ac, aa, bc, ca, ba\}, \{ab, ac, bb, cb\}, \{ac, ab, ca, ba\}\big\}$. The singleton sets are $\{bb\}, \{ca\}$ and thus $FS_2 = \{bb, ca\}$. It is easy to verify that for all $P \in P_{=2}(\mathcal{P}(F_A))$, $P$ has a nonempty intersection with $FS_2$. The algorithm terminates and outputs $\{bb, ca\}\}$, which are the forbidden subsequences which describe this language.

In sum, this procedure tells us that this language is SP for $k = 2$. Together DSP and FIND-SSQ provide a means to check whether a regular language is SP, and if it is to find the finite set of the shortest forbidden subsequences.

What is the time complexity for DSP and FIND-SSQ? Letting $n$ be the size of the syntactic monoid, the wholly nonzero condition can be checked with time

$O(n)$ and right annihilating condition runs in time $O(n^2)$. Thus DSP runs in $O(n^2)$. FIND-SSQ runs at time $O\big(p \cdot \mathrm{card}(\Sigma)\big)$, where $p$ is the total number of paths that leads to 0. Holzer studies the size of the syntactic monoid as a natural measure of descriptive complexity for regular languages [8].

## 6 Conclusion

Strictly Piecewise languages are wholly nonzero and right annihilating. The wholly nonzero property is shared by the Strictly Local languages and provides a definition for the "Strict" aspect, independent of the relation to the Testable classes. Also, the algebraic characterization for SP provides a polynomial-time decision procedure for a regular language in the size of its syntactic monoid. This paper also leaves open some interesting questions. In particular, we would like to know whether every wholly nonzero, $\mathcal{J}$-trivial language is right annihilating.

## References


1. Anderson, J.A.: Automata Theory with Modern Applications. Cambridge University Press (2006)
2. Clifford, A.: The Algebraic Theory of Semigroups. American Mathematical Society, Providence (1967)
3. García, P., Ruiz, J.: Learning k-testable and k-piecewise testable languages from positive data. Grammars 7, 125–140 (2004)
4. Green, J.A.: On the structure of semigroups. The Annals of Mathematics 54(1), pp. 163–172 (1951)
5. Haines, L.H.: On free moniods partially ordered by embedding. Journal of Combinatorial Theory 6, 94–98 (1969)
6. Heinz, J.: Learning long-distance phonotactics. Linguistic Inquiry 41(4), 623–661 (2010)
7. Higman, G.: Ordering by divisibility in abstract algebras. Proceedings of the London Mathematical Society 3(2), 326–336 (1952)
8. Holzer, M., König, B.: Regular languages, sizes of syntactic monoids, graph colouring, state complexity results, and how these topics are related to each other. EATCS Bulletin 83, 139–155 (June 2004)
9. Hopcroft, J.E.: An n log n algorithm for minimizing states in a finite automaton. Tech. rep., Stanford, CA, USA (1971)
10. McNaughton, R., Papert, S.: Counter-Free Automata. MIT Press (1971)
11. Pin, J.É., et A. Salomaa (éd.), G.R.: Syntactic semigroups, vol. 1. Springer Verlag (1997)
12. Rogers, J., Heinz, J., Bailey, G., Edlefsen, M., Visscher, M., Wellcome, D., Wibel, S.: On languages piecewise testable in the strict sense. In: Ebert, C., Jäger, G., Michaelis, J. (eds.) The Mathematics of Language. Lecture Notes in Artifical Intelligence, vol. 6149, pp. 255–265. Springer (2010)
13. Simon, I.: Piecewise testable events. In: Automata Theory and Formal Languages, pp. 214–222 (1975)
14. Thierrin, G.: Convex languages. In: ICALP'72. pp. 481–492 (1972)
15. Watanabe, T., Nakamura, A.: On the transformation semigroups of finite automata. Journal of Computer and System Sciences 26(1), 107–138 (1983)