Available online at www.sciencedirect.com

ScienceDirect

Discrete Applied Mathematics 155 (2007) 989–1006

DISCRETE APPLIED MATHEMATICS

www.elsevier.com/locate/dam

Recognizing splicing languages: Syntactic monoids and simultaneous pumping

Elizabeth Goode[a], Dennis Pixton[b,1]

[a]*Mathematics Department, Towson University, Towson, MD 21252, USA*
[b]*Department of Mathematical Sciences, Binghamton University, Binghamton, NY 13902-6000, USA*

Received 21 February 2004; received in revised form 16 October 2006; accepted 20 October 2006
Available online 8 December 2006

Abstract

We use syntactic monoid methods, together with an enhanced pumping lemma, to investigate the structure of splicing languages. We obtain an algorithm for deciding whether a regular language is a *reflexive* splicing language, but the general question remains open.

Keywords: Splicing systems; Splicing languages; Reflexive splicing languages; DNA splicing

1. Introduction

Tom Head [9] introduced the notion of *splicing* in formal language theory as a model for certain types of biochemical operations on DNA. In his formulation there is an initial language representing an initial set of double-stranded DNA (dsDNA) and a set of *splicing rules* that model the action of enzymes that cut and paste the dsDNA. The smallest language containing the initial language and closed under application of the splicing rules is called the *splicing language*. This setup has been codified and generalized, and is now known as an *H system*, see [11]. Throughout this paper we shall consider only finite H systems, with a finite set of rules and a finite initial language.

There have been several extensions of Head's original definitions. Throughout this paper we use the definitions due to Păun [11]. Specifically, a *splicing rule* is a 4-tuple $\langle u, u', v', v\rangle$ of strings, which we can use to *splice* two strings $xuu'y'$ and $x'v'vy$ at the indicated *sites* uu' and $v'v$ to produce the string $xuvy$.

Head's original definitions implicitly incorporated *reflexivity* and *symmetry* (see Section 4). These conditions are necessary for an accurate biological representation of DNA splicing systems: they both are consequences of the idea that the only requirement for recombination of strands of dsDNA is correct Watson–Crick complementarity. These extra conditions are lost in Păun's definition of splicing so we shall be explicit when we need reflexivity or symmetry. In more recent work Păun et al. [14] use the term 2-*splicing* to indicate symmetry assumptions, and many authors assume symmetry as part of the definition. See [1,4] for further comparison of splicing definitions, including a discussion of another extension due to Pixton.

[1] Research partially supported by NSF Grant #CCR-9509831.
E-mail addresses: egoode@towson.edu (E. Goode), dennis@math.binghamton.edu (D. Pixton).

0166-218X/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2006.10.006

One of Head's original problems was to determine the class of languages that arise as splicing languages. Culik and Harju [6] quickly proved that splicing languages are regular; their result was reproved in [16] and generalized in [17]. On the other hand, Gatterdam [7] produced the simple example $(aa)^*$ of a regular language that is not a splicing language. The precise characterization of splicing languages within regular languages remains unknown. There are related results by Bonizzoni et al. [2,3].

The main impetus for this paper is [10], in which Head exploited the connection between constants (see Section 7) and reflexive splicing languages. Our main result in this paper (see Section 6) is an algorithm for determining whether a given regular language L is the splicing language determined by a reflexive H system.

In Section 5 we adapt Head's main theorem to prove a characterization theorem for reflexive splicing languages. The point of the characterization is that we do not need to consider iterated splicing; nor do we need to explicitly provide an initial language. Our approach, rather, is to produce a finite set of reflexive splicing rules that can be used to generate a given language if in fact such a rule set exists. Indeed there is no obvious limit on the necessary size of such a rule set. We demonstrate that there is a limit with regard to rule set size in Section 6. We calculate this bound and by so doing we introduce the final ingredient in our algorithm for detecting reflexive splicing languages.

Our detection of reflexive splicing languages is algorithmic, and the key to our algorithm is determining an upper bound on the size of a splicing system that can generate L. While we do not give a "conceptual" characterization of such languages in this paper, we present numerous examples that shed light on the significance of our results, and pose questions that point to the challenge of developing such a characterization.

Bonizzoni et al. [4] have proved a characterization of reflexive splicing languages which is equivalent to our Theorem 5.2. Their result gives a more explicit form for the structure of reflexive splicing languages, which might well be useful in improving our detection algorithm.

In order to develop our algorithm we first introduce the notion of a "tuple language". A tuple language is a subset of $(A^*)^k$ to which we can apply formal language techniques via an identification of $(A^*)^k$ with the set of strings over the augmented language $A \cup \{\#\}$ which contains exactly $k-1$ copies of the separator #. The elementary facts of this approach are covered in Section 2.

Our use of tuple languages is very simple, and is mainly for ease of exposition. For an example of a much more thorough approach see, for example, Culik [5].

We express the splicing operation in terms of tuple languages, and by so doing we are able to establish a fundamental fact in Section 4: the set of splicing rules which leave a given regular language invariant is itself regular. This is important because a splicing language is of course invariant under splicing with its rule set.

In addition to introducing tuple languages in this paper, we introduce novel applications of several tools to the problem of characterizing reflexive splicing languages. One of the main tools is the syntactic monoid. The other main tool is Pixton's generalization of the pumping lemma for regular languages which we call the "simultaneous pumping lemma" or SPL. The SPL allows us to pump the same string in several different regular languages simultaneously. A proof of this lemma using the notion of tuple languages is presented in Section 3. We believe that the SPL will stand on its own and that it is applicable to formal language theory in general.

In Section 7 we revisit Head's paper [10]. Head's main result is that if there is a finite set of constants F for the regular language L so that $L \backslash A^* F A^*$ is finite, then L is a reflexive splicing language. We say such languages are finitely based on constants, or *FBC*, and we give a short proof of Head's original result based on our characterization theorem from Section 5. We then present another application of our detection methods, namely an algorithm to determine if a given regular language L has such a set of constants. Thus we answer the question Head posed in [10].

Finally, in Section 8 we present a number of examples to illustrate the differences between different types of splicing languages. These examples demonstrate that our results concerning reflexive splicing languages are specific within that class. In particular, we give examples demonstrating that not all splicing languages generated by finite H systems are reflexive, and that some are symmetric while others are not.

Many of the results presented in this paper were addressed in the first author's Ph.D. dissertation [8], although in most cases those that were proved there are given different proofs here. In particular, all results are now unified within the context of the tuple language approach using the syntactic monoid and the SPL. Further, within this unified context we have clarified the nature of the open questions still at hand.

Some of the results in this paper were announced at the DNA8 Workshop in Sapporo, see [12].

2. The syntactic monoid and tuple languages

In this section we review the syntactic monoid and specialize the basic definitions and results to tuple languages. Basic facts about the syntactic monoid are covered in many texts on formal language theory; Pin's book [15] presents a development of formal language theory in which the syntactic monoid plays a central role.

Throughout the paper A denotes a finite non-empty alphabet. We write 1 for the empty word in A^*.

If $L \subseteq A^*$ then we define *syntactic congruence* $\equiv_L$ *with respect to L as follows*:

$w \equiv_L z$ means that for all $x, y \in A^*$, xwy is in L if and only if xzy is in L.

A useful way to think of this is as follows: The *L-context* of a string w is the set of pairs $\langle x, y\rangle \in (A^*)^2$ satisfying $xwy \in L$. Then $w \equiv_L z$ means that w and z have the same L-context.

This relation is a congruence relation on A^*, so the quotient $A^*/\equiv_L$ is a monoid Syn L, called the *syntactic monoid* of L. The equivalence class of a string w in this quotient will sometimes be denoted by $[w]_L$; these equivalence classes are called *syntactic classes* (*with respect to L*). We write $\eta_L : A^* \to \text{Syn } L$ for the quotient homomorphism, which maps w to $[w]_L$.

The following facts are well known.

Theorem 2.1. *A language L is regular if and only if* Syn L *is finite. Moreover, if L is regular then each syntactic class* $[w]_L$*, for* $w \in A^*$*, is regular.*

We shall need the following notions. An *n-tuple of strings*, or more simply, an *n-tuple*, is an element of $(A^*)^n$, and an *n-tuple language* is a subset of $(A^*)^n$. If w is an n-tuple then we generally reserve the notation w_k for the kth component of w, so $w = \langle w_1, w_2, \ldots, w_n\rangle$. Note that all the tuples in a tuple language have the same number of components. As usual we identify $(A^*)^1$ with A^*; thus a language over A is the same as a 1-tuple language.

Next we select a symbol # which is not in A and we define $\bar{A} = A \cup \{\#\}$. We associate to an n-tuple w the *stringification* $s(w) = w_1 \# w_2 \# w_3 \# \cdots \# w_n$ in $\bar{A}^*$. In fact, stringification is a bijection from $(A^*)^n$ onto the set of words in $\bar{A}^*$ which contain exactly $n-1$ copies of #. Notice that $s(w) = w$ for $w \in A^*$. Using this bijection we can adapt the usual notions of formal language theory to the context of tuple languages. For example, we say a tuple language T is *regular* iff $s(T)$ is regular.

We would now like to specialize the notion of syntactic monoid to tuple languages T. We do not want to simply use Syn $s(T)$ for this purpose since most of our applications will not involve the separator symbol #, but just strings in A^*. To this end we make the following definitions.

If T is an n-tuple language and w and z are strings in A^* then we write $w \equiv_T z$ to mean $w \equiv_{s(T)} z$. In other words, for all $x, y \in \bar{A}^*$ we have $xwy \in s(T)$ if and only if $xzy \in s(T)$. For such a pair x, y suppose x contains j copies of # and y contains k copies of #. Remembering that w and z do not contain #, we may restrict x and y so that $j + k = n - 1$, since if $j + k \neq n - 1$ then neither xwy nor xzy can be in T. So we can rewrite the definition in terms of tuples as follows:

$$w \equiv_T z \text{ iff for all } p \text{ between 1 and } n \quad \text{and for all } x_1, x_2, \ldots, x_p, y_p, y_{p+1}, \ldots, y_n \in A^*,$$

$$\langle x_1, \ldots, x_{p-1}, x_p w y_p, y_{p+1}, \ldots, y_n\rangle \in T \iff \langle x_1, \ldots, x_{p-1}, x_p z y_p, y_{p+1}, \ldots, y_n\rangle \in T.$$

It is easy to check that $\equiv_T$ is a congruence relation on A^*, so we can define the syntactic monoid Syn $T = A^*/\equiv_T$ and the quotient homomorphism $\eta_T : A^* \to \text{Syn } T$ just as before. We shall also sometimes use the notation $[w]_T$ for the equivalence class of w in Syn T, and we shall refer to these classes as syntactic classes (with respect to T).

Theorem 2.2. *A tuple language T is regular if and only if* Syn T *is finite. Moreover, if T is regular then each syntactic class* $[w]_T$*, for* $w \in A^*$*, is regular.*

Proof. Suppose T is an n-tuple language.

For strings $w, z \in A^*$ we have by definition that $w \equiv_T z$ if and only if $w \equiv_{s(T)} z$, so $[w]_T = [w]_{s(T)}$ as subsets of A^*. Then the second part of the theorem follows immediately from Theorem 2.1 applied to $s(T)$.

Also, $[w]_T = [w]_{s(T)}$ provides a natural embedding of Syn T into Syn $s(T)$, so Syn T is finite if Syn $s(T)$ is finite. On the other hand, consider $w \in \bar{A}^*$. If w has more than $n-1$ copies of # then w cannot be a factor of a word of $s(T)$

so $[w]_{s(T)}$ is the zero element of Syn $s(T)$. Otherwise $[w]_{s(T)}$ can be written as a product $x_1\bar{\#}x_2\bar{\#}\cdots\bar{\#}x_k$ where $k\leqslant n$, $x_j \in \mathrm{Syn}\, T$ for $1\leqslant j\leqslant k$, and $\bar{\#} = [\#]_{s(T)}$. It follows that Syn $s(T)$ is finite if Syn T is finite.

So Syn T is finite if and only if Syn $s(T)$ is finite. But, according to Theorem 2.1, $s(T)$, and hence T, is regular if and only if Syn $s(T)$ is finite. So we have established the first part of the theorem. □

If x is a j-tuple and y is a k-tuple then we can interpret the pair $\langle x, y\rangle$ as the $(j+k)$-tuple $\langle x_1, \ldots, x_j, y_1, \ldots, y_k\rangle$. More generally, if x_k is an m_k-tuple then we can interpret $\langle x_1, x_2, \ldots, x_n\rangle$ as an m-tuple, with $m = m_1 + \cdots + m_n$. Conversely, any m-tuple may be reorganized in the form $\langle x_1, x_2, \ldots, x_n\rangle$ where x_k is an m_k-tuple. In this way we can consider the product $T_1 \times T_2 \times \cdots \times T_n$ of tuple languages to be a tuple language.

Lemma 2.3. *Suppose T_k is a non-empty m_k-tuple language for $1\leqslant k\leqslant n$ and let $T = T_1 \times T_2 \times \cdots \times T_n$. Then*:

(1) *For $w, z \in A^*$, $w\equiv_T z$ if and only if $w\equiv_{T_k} z$ for all k, $1\leqslant k\leqslant n$.*
(2) *The diagonal map $w \mapsto \langle w, w, \ldots, w\rangle$ of A^* to $(A^*)^n$ induces a natural injective homomorphism of* Syn T *into the direct product* Syn $T_1 \times$ Syn $T_2 \times \cdots \times$ Syn T_n.

Proof. Part (1): First suppose $w\equiv_T z$ and $1\leqslant k\leqslant n$. Suppose x and y are tuples of strings such that $s(x)ws(y) \in s(T_k)$. For each $j \neq k$ select $w_j \in T_j$ and let $X = \langle w_1, \ldots, w_{k-1}, x\rangle$ and $Y = \langle y, w_{k+1}, \ldots, w_n\rangle$, interpreted as tuples of strings. Then $s(X)ws(Y)$ is in $s(T)$, so $s(X)zs(Y)$ is in $s(T)$. But $s(X)zs(Y) = s(w_1)\#\cdots\#s(w_{k-1})\#s(x)zs(y)\#s(w_{k+1})\#\cdots\#s(w_n)$ and $s(T) = s(T_1)\#\cdots\#s(T_n)$, and we conclude, by counting #'s, that $s(x)zs(y)$ is in $s(T_k)$. So $s(x)ws(y) \in s(T_k)$ implies that $s(x)zs(y) \in s(T_k)$, and the reverse implication is proved by interchanging w and z. Therefore $z\equiv_{T_k} w$.

Conversely, suppose $w\equiv_{T_k} z$ for all k and suppose $s(X)ws(Y) \in s(T)$ for some tuples X and Y. Then we can interpret these tuples of strings as $X = \langle w_1, \ldots, w_{k-1}, x\rangle$ and $Y = \langle y, w_{k+1}, \ldots, w_n\rangle$ for some k, where each w_j is an m_j-tuple; thus,

$$s(X)ws(Y) = s(w_1)\#\cdots\#s(w_{k-1})\#s(x)ws(y)\#s(w_{k+1})\#\cdots\#s(w_n).$$

Using $s(T) = s(T_1)\#\cdots\#s(T_n)$ and counting #'s we conclude that each $s(w_j)$ is in $s(T_j)$ and that $s(x)ws(y) \in s(T_k)$. From $w\equiv_{T_k} z$ and $s(x)ws(y) \in s(T_k)$ we have $s(x)zs(y) \in s(T_k)$, and hence $s(X)zs(Y) = s(w_1)\#\cdots\#s(w_{k-1})\#s(x)zs(y)\#s(w_{k+1})\#\cdots\#s(w_n)$ is in $s(T)$. This shows that $s(X)ws(Y) \in s(T)$ implies that $s(X)zs(Y) \in s(T)$, and the reverse implication is proved by interchanging w and z. Therefore $z\equiv_T w$.

Part (2): One half of part (1) says that the induced map is well defined, and the other half says that it is an injection. It is easy to check that it is a homomorphism. □

Next we extend the notion of syntactic congruence to tuples, component-wise: if w and z are n-tuples and T is an m-tuple language then we write $w\equiv_T z$ to mean $w_j\equiv_T z_j$ for all j. This is purely a convenience for handling a number of syntactic congruences in parallel; it is *not* the same as the relation defined by $s(w)\equiv_{s(T)} s(z)$, which is much less useful. The following is the main reason for making this definition:

Lemma 2.4. *Suppose T is an n-tuple language and w and z are n-tuples. If $w\equiv_T z$ and $w \in T$ then $z \in T$.*

Proof. For $0\leqslant j\leqslant n$, write $z^j = \langle z_1, \ldots, z_j, w_{j+1}, \ldots, w_n\rangle$. Then $z^0 = w$ is in T. Inductively, assume $j > 0$ and z^{j-1} is in T. Write this as $z^{j-1} = \langle z_1, \ldots, z_{j-1}, 1w_j1, w_{j+1}, \ldots, w_n\rangle$. Applying $z_j\equiv_T w_j$ to this factorization we find that $z^j = \langle z_1, \ldots, z_{j-1}, 1z_j1, w_{j+1}, \ldots w_n\rangle \in T$. By induction, $z^n = z$ is in T. □

A version of the following "structure theorem" appears as [16, Lemmas 8.1–2] with a different proof:

Lemma 2.5. *An n-tuple language T is regular if and only if there is some $m\geqslant 0$ and there are regular languages T_{jk} for $1\leqslant j\leqslant m$ and $1\leqslant k\leqslant n$ so that*

$$T = \bigcup_{j=1}^{m} T_{j1} \times T_{j2} \times \cdots \times T_{jn}.$$

Proof. Suppose T is a regular n-tuple language. For $w \in (A^*)^n$ let $[w]_T$ be the equivalence class of w with respect to T. Since $\equiv_T$ is defined component-wise we obviously have $[w]_T = [w_1]_T \times [w_2]_T \times \cdots \times [w_n]_T$. Since T is regular there are finitely many such classes and each $[w_j]_T$ is regular, and Lemma 2.4 shows that T is a union of these classes.

The converse is easily proved using stringification. □

We shall routinely use syntactic congruence to show that certain tuple languages are regular, based on the following notion: we say a tuple language T *syntactically respects* a tuple language R iff for all $w, z \in A^*$, if $w \equiv_T z$ then $w \equiv_R z$. In other words, each syntactic class with respect to R is a union of syntactic classes with respect to T.

Lemma 2.6. *Suppose T is an n-tuple language and R is an m-tuple language. The following statements are equivalent*:

(1) *T syntactically respects R.*
(2) *For any $k > 0$ and all w and z in $(A^*)^k$, if $w \equiv_T z$ then $w \equiv_R z$.*
(3) *For all w and z in $(A^*)^m$, if $w \in R$ and $w \equiv_T z$ then $z \in R$.*

Proof. (1) implies (2): This is clear, since syntactic congruence on tuples is defined component-wise.

(2) implies (3): If $w \in R$ and $w \equiv_T z$ then $w \equiv_R z$. Hence $z \in R$ by Lemma 2.4.

(3) implies (1): Suppose w and z are in A^* and $w \equiv_T z$. Consider strings $x_1, \ldots, x_j, y_j, \ldots, y_m$ so that $\bar{w} = \langle x_1, \ldots, x_{j-1}, x_j w y_j, y_{j+1}, \ldots, y_m \rangle$ is in R. We have $\bar{w} \equiv_T \bar{z}$, where $\bar{z} = \langle x_1, \ldots, x_{j-1}, x_j z y_j, y_{j+1}, \ldots, y_m \rangle$, and so $\bar{z} \in R$. This demonstrates that $\langle x_1, \ldots, x_{j-1}, x_j w y_j, y_{j+1}, \ldots, y_m \rangle$ in R implies that $\langle x_1, \ldots, x_{j-1}, x_j z y_j, y_{j+1}, \ldots, y_m \rangle$ is in R, and reversing the roles of w and z in the argument gives the opposite implication. Thus $w \equiv_R z$. □

Lemma 2.7. *If T syntactically respects R then there is a natural surjective homomorphism from* Syn T *onto* Syn R. *If T is regular then so is R.*

Proof. We define φ: Syn $T \to$ Syn R by $[w]_T \mapsto [w]_R$; Lemma 2.6 shows that this definition is independent of the choice of w. It is easy to check that φ is a surjective homomorphism. Hence if T is regular then Syn T is finite, so Syn R is finite and R must be regular. □

3. Simultaneous pumping

Lemma 3.1. *Suppose L is a regular tuple language and let K be the cardinality of* Syn L; *let J be a positive integer. If w is a word in A^* with $|w| \geqslant JK$ then*:

(1) *There are $J + 1$ distinct prefixes of w which are syntactically congruent to each other.*
(2) *There are $J + 1$ distinct suffixes of w which are syntactically congruent to each other.*

Proof. Pigeonhole principle: Note that a string of length M has $M + 1$ distinct prefixes and $M + 1$ distinct suffixes. □

Theorem 3.2 (*The SPL*). *If $\mathscr{L}$ is a finite set of regular tuple languages then there is an integer n with the following property*:

If w is any word in A^* with length at least n then w can be factored as $w = \alpha\beta\gamma$ so that $\beta \neq 1$ and

$$\alpha\beta \equiv_L \alpha \quad \text{and} \quad \beta\gamma \equiv_L \gamma$$

for all L in $\mathscr{L}$.

Proof. If $\mathscr{L}$ contains only one language L we let K be the cardinality of Syn L and we let $n = K^2$. If $|w| \geqslant n$ then, by Lemma 3.1, w has $K + 1$ distinct prefixes $p_0, p_1, \ldots, p_K$ (ordered by size) which are equal in Syn L. Define s_j so that $w = p_j s_j$ and (by the pigeonhole principle) find $j < k$ so that $s_j \equiv_L s_k$. Then factor w as $p_j u s_k$ where $p_j u = p_k$ and $u s_k = s_j$. Define $\alpha = p_j$, $\beta = u$, and $\gamma = s_k$.

In the general case write $\mathscr{L} = \{L_1, L_2, \ldots, L_n\}$. We may delete the empty language if it appears in $\mathscr{L}$ since $z \equiv_\emptyset w$ is true for all z and w. We now apply the $n = 1$ case to the tuple language $T = L_1 \times L_2 \times \cdots \times L_n$. Lemma 2.3(1) allows us to interpret $z \equiv_T w$ as "$z \equiv_L w$ for all $L \in \mathscr{L}$". □

We shall be somewhat concerned with the size of the pumping length n in the SPL, so we define $\mathcal{N}(\mathcal{L})$ to be the smallest non-negative integer for which the SPL is true using $n = \mathcal{N}(\mathcal{L})$. We shall usually write $\mathcal{N}(L_1, \ldots, L_n)$ instead of $\mathcal{N}(\{L_1, \ldots, L_n\})$.

The following gives bounds on the size of $\mathcal{N}(\mathcal{L})$.

Theorem 3.3. (1) $\mathcal{N}(L) \leqslant K^2$ *where* K *is the cardinality of* Syn L.

(2) $\mathcal{N}(L_1, L_2, \ldots, L_n) \leqslant M^2$ *where* M *is the product of the cardinalities of* Syn L_k *for* $1 \leqslant k \leqslant n$.

(3) *If* $\mathcal{L}$ *and* $\mathcal{L}'$ *are two finite collections of regular languages and each* $L \in \mathcal{L}$ *is syntactically respected by some* $L' \in \mathcal{L}'$ *then* $\mathcal{N}(\mathcal{L}) \leqslant \mathcal{N}(\mathcal{L}')$.

Proof. Part (1): This is the bound used in the proof of the SPL.

Part (2): This follows from the embedding of Syn $(L_1 \times L_2 \times \cdots \times L_n)$ in the direct product of the monoids Syn L_k from Lemma 2.3(2).

Part (3): This is proved using the natural surjections of Lemma 2.7, which transform congruences $z \equiv_{L'} w$ into congruences $z \equiv_L w$. □

4. Rule sets

We shall have two uses for 4-tuples.

First, we consider a 4-tuple $r = \langle r_1, r_2, r_3, r_4 \rangle$ as a *splicing rule*. In this context we can *splice* two strings w_1 and w_2 using the rule r if we can factor $w_1 = u_1 r_1 r_2 u_2$ and $w_2 = u_3 r_3 r_4 u_4$, and in this case the result of the *splicing operation* is $z = u_1 r_1 r_4 u_4$. If L_0 is a language then we define $r(L_0)$ to be the set of such words z, where w_1 and w_2 range over L_0.

Now we can define an *H scheme* (also called a *splicing scheme*) to be a pair $\sigma = (A, R)$ where A is the alphabet and R is a 4-tuple language. We refer to the elements of R as the *rules* of σ; we say an H scheme is finite (or regular, etc.) if R is finite (or regular, etc.). We define the effect of an H scheme σ on a language L_0 as

$$\sigma(L_0) = \bigcup_{r \in R} r(L_0).$$

We define iterated splicing as follows:

$$\sigma^0(L_0) = L_0 \quad \text{and} \quad \sigma^{i+1}(L_0) = \sigma^i(L_0) \cup \sigma(\sigma^i(L_0)) \quad \text{for } i \geqslant 0,$$
$$\sigma^*(L_0) = \bigcup_{i \geqslant 0} \sigma^i(L_0).$$

Warning: In general, $\sigma^1(L_0) = L_0 \cup \sigma(L_0)$ is not equal to $\sigma(L_0)$. Similarly, we can have $\sigma^{i+1}(L) \neq \sigma(\sigma^i(L))$.

Note that $\sigma^*(L_0)$ is the smallest language in A^* which is closed under iterated splicing by σ and contains L_0. If $L = \sigma^*(L_0)$ for some *finite* H scheme σ and *finite* language L_0, we then say that L is a *splicing language*. (Languages defined by splicing using infinite rule sets have been considered in the literature, but throughout this paper we shall insist on finite initial languages and finite rule sets.)

The class of all splicing languages will be denoted by H. It is known that H is properly contained in the class of regular languages [11]. See Section 1 for some history and references.

In the rest of the paper we shall need to keep track of the exact splicing operations that generate a word, and for this reason we use a second interpretation of 4-tuples. We shall regard a 4-tuple q as a pair of factored words, $q_1 q_2$ and $q_3 q_4$, and we define the *spliced product* of q to be

$$\kappa(q) = q_1 q_4.$$

If Q is a 4-tuple language we define $\kappa(Q) = \{\kappa(q) : q \in Q\}$.

Lemma 4.1. *If* Q *is a regular* 4*-tuple language then* $\kappa(Q)$ *is regular.*

Proof. Starting with a representation $Q = \bigcup_{j=1}^{m} Q_{j1} \times Q_{j2} \times Q_{j3} \times Q_{j4}$ as in Lemma 2.5 we have $\kappa(Q) = \bigcup_{j=1}^{m} Q_{j1} Q_{j4}$. □

We are most interested in pairs of factorizations of words in a given language L, so we define $Q(L) = \{q \in (A^*)^4 : q_1q_2 \in L \text{ and } q_3q_4 \in L\}$.

Lemma 4.2. *L syntactically respects $Q(L)$, so $Q(L)$ is regular if L is regular.*

Proof. Suppose $q \in Q(L)$ and $q \equiv_L q'$. Then $q_1q_2 \in L$ and $q'_1q'_2 \equiv_L q_1q_2$, so $q'_1q'_2 \in L$. Similarly $q'_3q'_4 \in L$, so $q' \in Q(L)$. □

We need some definitions to help tie together these two uses of 4-tuples. These definitions will be used repeatedly in Sections 6 and 7.

Definition 4.3. The size of an n-tuple r is $|r| = \max\{|r_j| : 1 \leqslant j \leqslant n\}$.

Definition 4.4. For two 4-tuples r and $\bar{r}$ we write $r \preccurlyeq \bar{r}$ iff

(1) r_1 is a suffix of $\bar{r}_1$ and r_3 is a suffix of $\bar{r}_3$, and
(2) r_2 is a prefix of $\bar{r}_2$ and r_4 is a prefix of $\bar{r}_4$.

Definition 4.5. If R and Q are 4-tuple languages, we set

$$L(Q, R) = \{\kappa(q) : q \in Q \text{ and for some } r \in R, r \preccurlyeq q\},$$
$$L_N(Q, R) = \{\kappa(q) : q \in Q \text{ and for some } r \in R, r \preccurlyeq q \text{ and } |r| \leqslant N\}.$$

Then we have the following elementary translations:

Lemma 4.6. (1) *A word w is the result of splicing w_1 and w_2 using r if and only if there is a 4-tuple q so that $r \preccurlyeq q$, $q_1q_2 = w_1$, $q_3q_4 = w_2$, and $w = \kappa(q)$.*
(2) *If L_0 is a language and $\sigma = (A, R)$ is an* H *scheme then $\sigma(L_0) = L(Q(L_0), R)$.*

As an illustration of this translation, the following gives a proof of the well-known fact that single splicing preserves regularity.

Lemma 4.7. *If Q and R are regular then $L(Q, R)$ is regular. If Q is regular then $L_N(Q, R)$ is regular.*

Proof. First let $\bar{R} = \{\bar{r} \in (A^*)^4 : \text{for some } r \in R, r \preccurlyeq \bar{r}\}$. Using Lemma 2.5, $R = \bigcup_{j=1}^{m} R_{j1} \times R_{j2} \times R_{j3} \times R_{j4}$ where the languages R_{jk} are regular. Then $\bar{R} = \bigcup_{j=1}^{m} A^*R_{j1} \times R_{j2}A^* \times A^*R_{j3} \times R_{j4}A^*$, so $\bar{R}$ is regular. Since $L(Q, R) = \kappa(Q \cap \bar{R})$ we conclude that $L(Q, R)$ is regular. Finally, $L_N(Q, R) = L(Q, R_N)$ where R_N is the finite set $\{r \in R : |r| \leqslant N\}$, so $L_N(Q, R)$ is regular. □

We need a regularity result for sets of rules. If L is a language then we say a rule r *respects* L iff $r(L) \subseteq L$, and we define $R(L)$ to be the set of all rules that respect L. In other words, $R(L) = \{r \in (A^*)^4 : r(L) \subseteq L\}$.

Theorem 4.8. *L syntactically respects $R(L)$, and so $R(L)$ is regular if L is regular.*

Proof. Suppose $r \in R(L)$ and $r' \equiv_L r$. To show that $r' \in R(L)$ we take $q' \in Q(L)$ so that $r' \preccurlyeq q'$, and we need to show that $\kappa(q') \in L$.

We have $q' = \langle u_1r'_1, r'_2u_2, u_3r'_3, r'_4u_4\rangle$ for some strings u_j. Since $r'_j \equiv_L r_j$ for each j we have $q' \equiv_L q = \langle u_1r_1, r_2u_2, u_3r_3, r_4u_4\rangle$, so, by Lemma 4.2, $q \in Q(L)$. Then, since r respects L, $\kappa(q) = u_1r_1r_4u_4$ is in L. Finally, $\kappa(q) \equiv_L u_1r'_1r'_4u_4 = \kappa(q')$ so $\kappa(q') \in L$, as desired. □

We call $\langle r_1, r_2\rangle$ and $\langle r_3, r_4\rangle$ the *sites* of the rule r. We also, when the context demands strings rather than pairs, refer to r_1r_2 and r_3r_4 as the sites of r.

Now an H scheme $\sigma = (A, R)$ specifies a set R of pairs of sites, so it defines a relation on the set S of sites. We say σ is *reflexive* if R defines a reflexive relation on S, and we say it is *symmetric* if R defines a symmetric relation on S. More explicitly:

(1) σ is reflexive iff $r \in R$ implies that both $\dot{r} = \langle r_1, r_2, r_1, r_2\rangle$ and $\ddot{r} = \langle r_3, r_4, r_3, r_4\rangle$ are in R.
(2) σ is symmetric iff $r \in R$ implies $\hat{r} = \langle r_3, r_4, r_1, r_2\rangle \in R$.

We say σ is *reflexive–symmetric* if it is both reflexive and symmetric.

Given a language L we define corresponding subsets of $R(L)$:

(1) $r \in R^r(L)$ iff $r \in R(L)$ and both $\dot{r} = \langle r_1, r_2, r_1, r_2\rangle$ and $\ddot{r} = \langle r_3, r_4, r_3, r_4\rangle$ respect L.
(2) $r \in R^s(L)$ iff $r(L) \subseteq L$ and $\hat{r} = \langle r_3, r_4, r_1, r_2\rangle$ respects L.
(3) $R^{rs}(L) = R^r(L) \cap R^s(L)$.

Now we have an addendum to Theorem 4.8:

Theorem 4.9. *For t in the set $\{r, s, rs\}$, L syntactically respects $R^t(L)$, and so $R^t(L)$ is regular if L is regular.*

Proof. Suppose that r is in $R^r(L)$ and $\rho \equiv_L r$, so $\rho_j \equiv_L r_j$ for all j. Then $\dot{r} = \langle r_1, r_2, r_1, r_2\rangle \equiv_L \dot{\rho} = \langle \rho_1, \rho_2, \rho_1, \rho_2\rangle$ and $\ddot{r} = \langle r_3, r_4, r_3, r_4\rangle \equiv_L \ddot{\rho} = \langle \rho_3, \rho_4, \rho_3, \rho_4\rangle$. Since r, $\dot{r}$, and $\ddot{r}$ are all in $R(L)$, Theorem 4.8 implies that ρ, $\dot{\rho}$, and $\ddot{\rho}$ are all in $R(L)$, so $\rho \in R^r(L)$.

The same kind of argument covers the other two cases. □

We say a language L is a *reflexive splicing language* iff there is a finite reflexive H scheme σ and a finite language L_0 so that $L = \sigma^*(L_0)$. The class of such languages is denoted by H^r. We define similarly *symmetric* and *reflexive–symmetric* splicing languages and the classes H^s and H^{rs}.

Remark 4.10. Examples 8.3, 8.4, and 8.7 show that the inclusions $H^r \cup H^s \subseteq H$, $H^r \cap H^s \subseteq H^r$, and $H^r \cap H^s \subseteq H^s$ are proper. Obviously, $H^{rs} \subseteq H^r \cap H^s$, but we do not know whether this inclusion is proper. See Section 8 for examples.

Lemma 4.11. *Suppose $L \subseteq A^*$ and $t \in \{r, s, rs\}$. Then L is in the class H^t iff $L = \sigma^*(L_0)$ where L_0 is finite and $\sigma = (A, R)$ is a finite* H *scheme with $R \subseteq R^t(L)$.*

Proof. The three cases are similar; we give the argument for reflexive splicing languages.

If L is a reflexive splicing language then $L = \sigma^*(L_0)$ where L_0 is finite and $\sigma = (A, R)$ is a finite reflexive H scheme. Then $\sigma(L) \subseteq L$ so $R \subseteq R(L)$. Moreover, if r is in R then both $\dot{r} = \langle r_1, r_2, r_1, r_2\rangle$ and $\ddot{r} = \langle r_3, r_4, r_3, r_4\rangle$ are in R, so both $\dot{r}$ and $\ddot{r}$ are in $R(L)$. Hence r is in $R^r(L)$.

Conversely, suppose $L = \sigma^*(L_0)$ where L_0 is finite and $\sigma = (A, R)$ is a finite H scheme with $R \subseteq R^r(L)$. Let $\tilde{R}$ be the set of rules that are in R or have the form $\dot{r} = \langle r_1, r_2, r_1, r_2\rangle$ or $\ddot{r} = \langle r_3, r_4, r_3, r_4\rangle$ where r is in R. Then $\tilde{\sigma} = (A, \tilde{R})$ is a finite reflexive H scheme. Moreover, $\tilde{R} \subseteq R(L)$, and this implies that $\tilde{\sigma}(L) \subseteq L$. Since $L_0 \subseteq L$, we conclude that $\tilde{\sigma}^*(L_0) \subseteq L$. Also, $\tilde{R} \supset R$ implies that $\tilde{\sigma}^*(L_0) \supset \sigma^*(L_0) = L$. Thus $\tilde{\sigma}^*(L_0) = L$ and we have shown that L is a reflexive splicing language. □

5. Characterizing reflexive splicing languages

Lemma 5.1. *Suppose s is a site of a rule in $R^r(L)$. Then, for any strings u and v, the rules $r' = \langle usv, 1, usv, 1\rangle$ and $r'' = \langle 1, usv, 1, usv\rangle$ are in $R^{rs}(L)$.*

Proof. Suppose $s = r_1 r_2$ where $r \in R^r(L)$. Then $\dot{r} = \langle r_1, r_2, r_1, r_2\rangle$ is in $R(L)$. If $w_1 = x_1 usvy_1$ and $w_2 = x_2 usvy_2$ are in L then the result of splicing w_1 and w_2 using either r' or r'' is $z = x_1 usvy_2$. But the result of splicing $w_1 = x_1 ur_1 r_2 vy_1$ and $w_2 = x_2 ur_1 r_2 vy_2$ using $\dot{r}$ is $x_1 ur_1 r_2 vy_2 = x_1 usvy_2 = z$, so z is in L. Hence both $r'(L) \subseteq L$ and $r''(L) \subseteq L$. Since r' and r'' are also self-symmetric and self-reflexive they are in $R^{rs}(L)$. □

Theorem 5.2. *Suppose $t \in \{r, rs\}$ and L is a regular language. The following are equivalent*:

(1) *L is in the class H^t.*
(2) *There is a finite H scheme $\sigma_0 = (A, R_0)$ with $R_0 \subseteq R^t(L)$ so that $L \backslash \sigma_0(L)$ is finite.*

Moreover, if part (2) *is satisfied then we can represent $L = \sigma^*(L_0)$ where L_0 is finite and all rules of σ which are not in σ_0 have one of the forms $\langle u, 1, v, 1\rangle$ or $\langle 1, u, 1, v\rangle$.*

Remark 5.3. Theorem 5.2 is false if we omit the word "reflexive" and replace $R^r(L)$ with $R(L)$: see Example 8.8.

Proof of Theorem 5.2. Part (1) trivially implies part (2).

For the converse, suppose we are given L and σ_0 as in part (2). We shall find a finite H scheme $\sigma = (A, R)$ with $R \subseteq R^t(L)$ and a finite set L_0 so that $L = \sigma^*(L_0)$. This is enough, by Lemma 4.11.

Let N be the maximum length of a site of a rule of σ_0 and let K be the cardinality of Syn L. Define σ to consist of all rules of σ_0 plus all rules in $R^t(L)$ of size at most $2K + N$ of the forms $\langle u, 1, v, 1\rangle$ or $\langle 1, u, 1, v\rangle$, and let L_0 consist of $L \backslash \sigma_0(L)$ together with all words of L of length less than or equal to $4K + N$. Since all rules of σ are in $R^t(L)$ and $L_0 \subseteq L$ we have $\sigma^*(L_0) \subseteq L$.

So we only need to show that $L \subseteq \sigma^*(L_0)$. First consider the set L_1 of all words in L that contain a site of σ_0. Let w be a word of $L_1 \backslash L_0$. Then $w = xsy$ where s is a site of a rule of σ_0 and $|w| > 4K + N$, so either x or y has length greater than $2K$. We give the argument in case $|y| > 2K$. By Lemma 3.1 we can factor $y = tuvz$ where $t \equiv_L tu \equiv_L tuv$, neither u nor v is empty, and $|tuv| \leqslant 2K$. The rule $\bar{r} = \langle stuv, 1, stuv, 1\rangle$ is in $R^t(L)$ by Lemma 5.1. Let $r = \langle stu, 1, st, 1\rangle$. Since $stuv \equiv_L stu \equiv_L st$ we have $r \equiv_L \bar{r}$, so $r \in R^t(L)$ by Theorem 4.9 Also, $|r| < 2K + N$ so r is a rule of σ. Now consider $w_1 = xstuz$ and $w_2 = xstvz$; these both have length less than w. Since $tu \equiv_L tuv$ we have $w_1 \equiv_L xstuvz = w$; and since $t \equiv_L tu$ we have $tv \equiv_L tuv$, so $w_2 \equiv_L w$. Thus both w_1 and w_2 are in L. Moreover, w_1 and w_2 splice using r to give w. This is the inductive step in proving that $L_1 \subseteq \sigma^*(L_0)$.

To finish the proof, suppose $w \in L \backslash L_0$. Then $w \in \sigma_0(L)$, so there are two strings w_1 and w_2 of L which splice using a rule r_0 of σ_0 to give w. Then w_1 and w_2 contain sites of rules in σ_0, so they are in L_1, and r_0 is a rule of σ, so $w \in \sigma^*(L_0)$. □

One of the reviewers of this paper noticed the following interesting interpretation of Theorem 5.2:

Corollary 5.4. *If L is a regular language and $L = \sigma(L)$ for some finite reflexive H scheme σ then there is a finite subset L_0 of L so that $L = \sigma^*(L_0)$.*

6. Detecting reflexive splicing languages

Suppose Q and R are 4-tuple languages. We consider the increasing sequence of languages $L_N(Q, R)$ and their union $L(Q, R)$ as defined in Definition 4.5, and we consider the possibility that the sequence converges in the sense that it is eventually constant. Our main result is a "convergence theorem" which gives a limit on when such convergence must appear.

Theorem 6.1. *Suppose that Q and R are regular 4-tuple languages, and set $\bar{R} = \{\bar{r} \in (A^*)^4$: for some $r \in R$, $r \preccurlyeq \bar{r}\}$. Let $n = \mathcal{N}(Q, \bar{R})$ as provided by the SPL. Then $L_N(Q, R) = L(Q, R)$ for some N if and only if $L_{2n}(Q, R) = L(Q, R)$.*

Before we start the proof of Theorem 6.1 we make some simplifying observations.

We need the extended rule set $\bar{R}$ so we can give an explicit, a priori calculation of n. The proof of Lemma 4.7 shows that $\bar{R}$ is regular and it follows immediately from the definitions that $L_N(Q, \bar{R}) = L_N(Q, R)$ for all N, and $L(Q, \bar{R}) = L(Q, R)$. Hence, other than affecting the exact value of n, there is little difference between R and $\bar{R}$, and, in fact, $\bar{R} = R$ is true in our applications of Theorem 6.1.

So we shall simplify notation for the remainder of the proof by assuming that $R = \bar{R}$. This means that we are assuming

$$r \in R \quad \text{and} \quad r \preccurlyeq \bar{r} \Longrightarrow \bar{r} \in R.$$

The next observation is that the sizes of r_2 and r_3 is not an issue:

Lemma 6.2. *Suppose Q, R, and n are as in Theorem* 6.1. *Suppose $q \in Q$, $r \in R$, and $r \preccurlyeq q$. Then there are $\tilde{q} \in Q$ and $\tilde{r} \in R$ so that*

(1) $\tilde{r} \preccurlyeq \tilde{q}$;
(2) $\tilde{r}_j = r_j$ *and* $\tilde{q}_j = q_j$ *for* $j = 1, 4$;
(3) $|\tilde{r}_2| \leqslant n$ *and* $|\tilde{r}_3| \leqslant n$.

Proof. Suppose that $|r_2| > n$. Then we can factor $r_2 = \alpha\beta\gamma$ according to the SPL for the family $\{Q, R\}$. Since $r \preccurlyeq q$ we can factor $q_2 = r_2 u$. We define $r' = \langle r_1, r'_2, r_3, r_4\rangle$ and $q' = \langle q_1, q'_2, q_3, q_4\rangle$ where $r'_2 = \alpha\gamma$ and $q'_2 = r'_2 u$. Then $r' \preccurlyeq q'$, and we have $r' \in R$ and $q' \in Q$ since $r' \equiv_R r$ and $q' \equiv_Q q$. Moreover, $|r'_2| < |r_2|$ since $\beta \neq 1$.

A finite number of iterations of this process, or the similar process applied to the third component, produce the desired $\tilde{r}$ and $\tilde{q}$. □

Now we need to set up the basic induction which proves Theorem 6.1.

We need some terminology for the locations of various factors of a word w, and for this we define the *positions* of w, as follows. If $w = a_1 a_2 \ldots a_m$ with a_j in A then the set of positions of w is the set of integers $\{0, 1, \ldots, m\}$. We consider the positions to occur *between* the symbols of w, or at the ends of w. Thus, specifying a position p in w is equivalent to specifying a factorization $w = uv$, so that p is the position separating the factors u and v.

If p and p' are positions in w and $p \leqslant p'$ then we use the notation $[p, p']$ for the set of positions between p and p' (inclusive), and we refer to this set as a *segment* of w. We shall also interpret the segment $[p, p']$ as the substring $a_{p+1} \ldots a_{p'}$. Conversely, if a substring of w is specified, including its placement within w, we shall interpret the substring as a segment.

We now use this notion to count the number of ways a word can be generated by splicing. We define, for $w \in L(Q, R)$ and $k \geqslant 0$, a set $P_k(w)$ of positions as follows: a position p of w is in $P_k(w)$ if and only if there are tuples $q \in Q$ and $r \in R$ with $r \preccurlyeq q$, $|r| \leqslant k$, and $w = q_1 q_4$, so that p is the position separating the factors q_1 and q_4. Then $P_k(w)$ is finite, and it is non-empty if and only if $w \in L_k(Q, R)$.

Here, then, is the main induction step:

Lemma 6.3. *Suppose Q, R, and n are as in Theorem* 6.1. *Suppose $L_N(Q, R) = L(Q, R)$ with $N > 2n$ and suppose $\zeta \in L(Q, R) \backslash L_{N-1}(Q, R)$. Then there is $z \in L(Q, R) \backslash L_{N-1}(Q, R)$ so that $P_N(z)$ has smaller cardinality than $P_N(\zeta)$.*

We shall first verify that Lemma 6.3 implies Theorem 6.1:

Proof of Theorem 6.1. Suppose $L_N(Q, R) = L(Q, R)$ for some N, and let N_0 be the minimum integer for which $L_{N_0}(Q, R) = L(Q, R)$. If $N_0 > 2n$ then choose $\zeta \in L(Q, R) \backslash L_{N_0-1}(Q, R)$ so that the cardinality of $P_{N_0}(\zeta)$ is as small as possible. But then Lemma 6.3 applied to this ζ provides a contradiction. Hence we have $N_0 \leqslant 2n$, so $L_{2n}(Q, R) = L(Q, R)$.

The converse is trivial. □

So now all we have to do is prove the induction step:

Proof of Lemma 6.3. We have $L_N(Q, R) = L(Q, R)$ with $N > 2n$ and $\zeta \in L(Q, R) \backslash L_{N-1}(Q, R)$. We fix a position $m \in P_N(\zeta)$; the plan is to produce $z \in L(Q, R) \backslash L_{N-1}(Q, R)$ by removing m without introducing any new positions in $P_N(z)$. We have the following starting configuration.

Claim 6.4. *There are $Z \in Q$ and $\rho \in R$ so that*

(1) $Z_1 Z_4 = \zeta$, $\rho \preccurlyeq Z$, *and m is the position between Z_1 and Z_4.*
(2) $|\rho_2| < N$, $|\rho_3| < N$, *and either* $|\rho_1| = N$ *or* $|\rho_4| = N$.
(3) $|\rho_1| < n$ *implies that* $\rho_1 = Z_1$ *and* $|\rho_4| < n$ *implies that* $\rho_4 = Z_4$.

Proof. Here the existence of Z and ρ and part (1) are clear by definition of $L(Q, R)$. We can choose ρ so that $|\rho| \leqslant N$ because $\zeta \in L_N(Q, R)$, but $|\rho| < N$ is false since $\zeta \notin L_{N-1}(Q, R)$. We can arrange $|\rho_2| < N$ and $|\rho_3| < N$ by Lemma 6.2, so either $|\rho_1| = N$ or $|\rho_4| = N$, establishing part (2). Now suppose $|\rho_1| < n$. If $\tilde{\rho}_1$ is any suffix of Z_1 which contains ρ_1 then $\tilde{\rho} = \langle \tilde{\rho}_1, \rho_2, \rho_3, \rho_4 \rangle$ is in R since $\rho \preccurlyeq \tilde{\rho}$, and clearly $\tilde{\rho} \preccurlyeq Z$. Hence, we may replace ρ by $\tilde{\rho}$ to ensure that $\rho_1 = Z_1$ if $|Z_1| < n$, and $|\rho_1| = n$ otherwise. This, with the symmetrical consideration for ρ_4, proves part (3). □

We now construct z from ζ. This requires that we "inflate" certain substrings of ζ as described next. We shall consider m as a "middle position" in ζ dividing ζ into the two halves Z_1 and Z_4, and we shall treat these two halves symmetrically.

If $|\rho_1| \geqslant n$ we let δ_1 be the suffix of ρ_1 of length n, and we factor this as $\delta_1 = \alpha_1\beta_1\gamma_1$ according to the SPL for the family $\{Q, R\}$. Alternatively, if $|\rho_1| < n$ we have $Z_1 = \rho_1$, and we define $\delta_1 = \rho_1$, but we do not define α_1, β_1, or γ_1. We define δ_4 symmetrically, as the prefix of ρ_4 of length n if $|\rho_4| \geqslant n$ and as $\delta_4 = \rho_4$ otherwise; and in the first case we factor $\delta_4 = \alpha_4\beta_4\gamma_4$ according to the SPL for $\{Q, R\}$.

We define an operation of "inflation" on segments w of ζ as follows: if w contains the segment $\alpha_1\beta_1\gamma_1$ as described above then we replace the segment β_1 with β_1^2; and if w contains the segment $\alpha_4\beta_4\gamma_4$ then we replace the segment β_4 with β_4^2.

We define $z = \zeta^{(2)}$. Informally, we are just squaring the segments β_1 and β_2 if they occur in ζ. At least one segment is duplicated in this way, and at most two are duplicated. See the diagrams below.

We need to show that $z \in L(Q, R) \backslash L_{N-1}(Q, R)$ and that $P_N(z)$ has smaller cardinality than $P_N(\zeta)$.

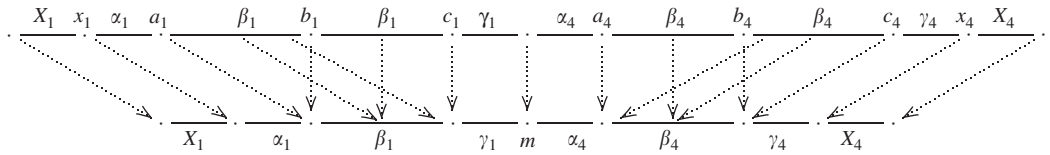
Claim 6.5. *Define* $Z^{(2)} = \langle Z_1^{(2)}, Z_2, Z_3, Z_4^{(2)} \rangle$ *and* $\rho^{(2)} = \langle \rho_1^{(2)}, \rho_2, \rho_3, \rho_4^{(2)} \rangle$. *Then* $Z^{(2)} \in Q$, $\rho^{(2)} \in R$, $\rho^{(2)} \preccurlyeq Z^{(2)}$, *and* $\kappa(Z^{(2)}) = z$. *Hence* $z \in L(Q, R)$.

Proof. Note that the SPL implies that $w^{(2)} \equiv_Q w$ and $w^{(2)} \equiv_R w$ for any segment w of ζ. Hence $Z^{(2)} \equiv_Q Z$, so $Z^{(2)} \in Q$ by Lemma 2.4. Similarly $\rho^{(2)} \equiv_R \rho$, so $\rho^{(2)} \in R$. It is immediate that $\rho^{(2)} \preccurlyeq Z^{(2)}$, and $\kappa(Z^{(2)}) = z$. □

Since we shall concentrate on the strings δ_1 and δ_4 we factor ζ as $X_1\delta_1\delta_4X_4$, with a corresponding factorization for z as $X_1\delta_1^{(2)}\delta_4^{(2)}X_4$.

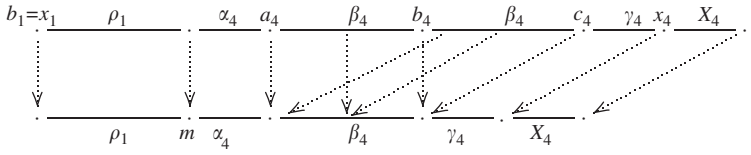
We need to examine the positions p in $P_N(z)$. For this we define a mapping π from the positions of z to the positions of ζ. This mapping has the effect of "deflating" various segments of z. This mapping is best described by the following diagrams.

In the first diagram we show the mapping in case both ρ_1 and ρ_4 have length $\geqslant n$. We have indicated various positions a_i, b_i, c_i, and x_i in z (for $i = 1$ or 4) that we shall use in the discussion below, as well as the middle position m in ζ.

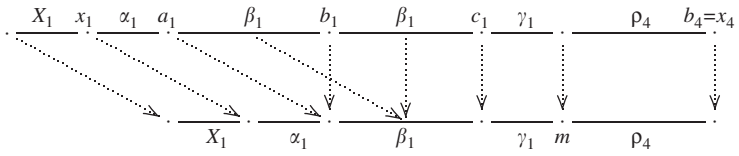


Note that π is piece-wise monotone, so it is just a translation on the positions left of b_1, on the positions from b_1 to b_4, and on the positions to the right of b_4. All positions of the factor β_1^2 of z map to the corresponding positions of β_1 in w, except that the ambiguity at b_1 is resolved in favor of the left endpoint of β_1. The description of the mapping on β_4^2 is just the mirror image.

If ρ_1 has length less than n then the diagram for π is modified as below. The X_1 factor is empty, since $\rho_1 = Z_1 = Z_1^{(2)}$, and there is no β_1 segment to be doubled. We do not define a_1 or c_1 in this case, but it is convenient to set $b_1 = x_1$.



In case ρ_4 has length less than n we have the following mirror image diagram:



If $s = [j, k]$ is a segment in z then we define $\pi(s) = [\pi(j), \pi(k)]$, provided that $\pi(j) \leqslant \pi(k)$. *Caution*: This is not always the same as $\{\pi(x): x \in s\}$, so "obvious" statements like "$s \subseteq t \implies \pi(s) \subseteq \pi(t)$" may not be true.

Claim 6.6. *Suppose s is a segment of z which is not contained in the interior of either $[a_1, c_1]$ or $[a_4, c_4]$. Then*:

(1) *$\pi(s)$ is defined, and $|\pi(s)| \leqslant |s|$.*
(2) *$|\pi(s)| < |s|$ if s contains $[a_1, b_1]$ or $[b_4, c_4]$.*
(3) *If s is disjoint from both $[a_1, b_1]$ and $[b_4, c_4]$ or s contains either $[x_1, a_1]$ or $[c_4, x_4]$ then $\pi(s) \equiv_Q s$ and $\pi(s) \equiv_{\bar{R}} s$.*

Remark 6.7. The *interior* of a segment $[p, p']$ is just $[p, p']\backslash\{p, p'\}$. Also, if $|\delta_1| < n$ then a_1 and c_1 do not exist, and all statements involving them in the lemma should be removed, and similarly if $|\delta_4| < n$.

Proof. Part (1): It is easy to check that $\pi(s)$ is defined using the fact that π preserves order except on the interiors of $[a_1, c_1]$ and $[a_4, c_4]$. Either $\pi(s)$ is equal to s (as a string) or is obtained from s by deleting a copy of β_1 or β_4 or both, so $|\pi(s)| \leqslant |s|$.

Part (2): If s contains either $[a_1, b_1]$ or $[b_4, c_4]$ then the corresponding copy of β_1 or β_4 is erased in $\pi(s)$.

Part (3): If s is disjoint from both $[a_1, b_1]$ and $[b_4, c_4]$ then $\pi(s) = s$ (as strings).

Suppose s contains $[x_1, a_1]$; then β_1 is defined. If s contains $[x_1, a_1]$ but not $[a_1, b_1]$ then again $\pi(s) = s$. Alternatively, s contains $[x_1, b_1]$ so, as strings, s contains $\alpha_1\beta_1$ and this copy of β_1 is erased in $\pi(s)$. If β_4 is also defined and β_4 is erased in $\pi(s)$ then s contains the segment $[a_4, b_4]$ so, as strings, s contains $\alpha_4\beta_4$. But $\alpha_1\beta_1 \equiv_Q \alpha_1$ and $\alpha_4\beta_4 \equiv_Q \alpha_4$ by the SPL, and similarly for congruence with respect to $\bar{R}$, so s and $\pi(s)$ are congruent with respect to both Q and $\bar{R}$.

The symmetric argument holds if we start by assuming that s contains $[a_4, x_4]$. □

Now consider $p \in P_N(z)$, and select corresponding $q \in Q$ and $r \in \bar{R}$ with $r \preccurlyeq q$, $|r| \leqslant N$, $z = q_1q_4$, so that p is the position separating the factors q_1 and q_4. We shall investigate the relationship between p and the position $\pi(p)$ in ζ.

First we adjust r so that Claim 6.6 will apply. By Lemma 6.2 we may assume r_2 and r_3 have length less than N. We define $\bar{r}_1 = r_1$ unless r_1 is contained in $[x_1, x_4]$; in this case we define $\bar{r}_1 = [x_1, p]$. Since p is the right endpoint of r_1 we see that r_1 is a suffix of $\bar{r}_1$ and $\bar{r}_1$ is a suffix of q_1. Similarly, we define $\bar{r}_4 = r_4$ unless r_4 is contained in $[x_1, x_4]$, in which case $\bar{r}_4 = [p, x_4]$. If we let $\bar{r} = \langle \bar{r}_1, r_2, r_3, \bar{r}_4\rangle$ then we have $r \preccurlyeq \bar{r} \preccurlyeq q$. Now $\bar{r}_1$ and $\bar{r}_4$ satisfy the assumptions of Claim 6.6, as do q_1 and q_4, so their images under π are defined.

Claim 6.8. *Define the rules $\bar{r} = \langle \bar{r}_1, r_2, r_3, \bar{r}_4\rangle$ and $\tilde{r} = \langle \pi(\bar{r}_1), r_2, r_3, \pi(\bar{r}_4)\rangle$, and the quadruple $\tilde{q} = \langle \pi(q_1), q_2, q_3, \pi(q_4)\rangle$. Then*:

(1) *$\tilde{r} \in \bar{R}$, $\tilde{q} \in Q$, $\tilde{r} \preccurlyeq \tilde{q}$, $\tilde{q}_1\tilde{q}_4 = \zeta$, and $\pi(p)$ is the position separating these two factors.*
(2) *$|\tilde{r}| \leqslant N$.*
(3) *$|\tilde{r}| < N$ if $|r| < N$ or $p \in [b_1, b_4]$.*

Proof. Part (1): Claim 6.6(3) applies to show that $\tilde{q}_i = \pi(q_i) \equiv_Q q_i$ for $i = 1, 4$. Then $\tilde{q} \equiv_Q q$ and, since $q \in Q$, we have $\tilde{q} \in Q$ by Lemma 2.4. Next, since $r \preccurlyeq \bar{r}$ we have $\bar{r} \in \bar{R}$. We have adjusted $\bar{r}_1$ and $\bar{r}_4$, if necessary, so that Claim 6.6(3) applies, so $\tilde{r}_i = \pi(\bar{r}_i) \equiv_{\bar{R}} \bar{r}_i$ for $i = 1, 4$. Hence $\tilde{r} \equiv_{\bar{R}} \bar{r}$,and so $\tilde{r} \in \bar{R}$. The rest of part (1) is easy to check.

Part (2): If r_1 is contained in $[x_1, x_4]$ then $\bar{r}_1 = [x_1, p]$ so $\tilde{r}_1 = \pi(\bar{r}_1)$ is contained in $[\pi(x_1), \pi(x_4)]$, which equals $\delta_1\delta_4$ as a string. So $|\tilde{r}_1| \leqslant |\delta_1| + |\delta_4| \leqslant 2n < N$. If r_1 is not contained in $[x_1, x_4]$ then $|\tilde{r}_1| = |\pi(r_1)| \leqslant |r_1| \leqslant N$ by Claim 6.6(1). The same considerations apply to $\tilde{r}_4$, so we have $|\tilde{r}| \leqslant N$.

Part (3): If r_1 is contained in $[x_1, x_4]$ then, as above, $|\tilde{r}_1| \leqslant 2n < N$. Otherwise $\bar{r}_1 = r_1$. If $|r_1| < N$ then $|\tilde{r}_1| = |\pi(r_1)| \leqslant |r_1| < N$, and if $p \in [b_1, b_4]$ then $\bar{r}_1 = r_1$ contains $[x_1, b_1]$ so, by Claim 6.6(2), $|\tilde{r}_1| = |\pi(r_1)| < |r_1| \leqslant N$. Thus, in either case, $|\tilde{r}_1| < N$. Similarly $|\tilde{r}_4| < N$, so $|\tilde{r}| < N$. □

We now list several immediate consequences of Claim 6.8:

(1) π maps $P_N(z)$ into $P_N(\zeta)$: This is just Claim 6.8, parts (1) and (2).
(2) $z \notin L_{N-1}(Q, R)$: If so we can find $p \in P_{N-1}(z)$. But then $|\tilde{r}| < N$, so $\pi(p) \in P_{N-1}(\zeta)$, so $\zeta \in L_{N-1}(Q, R)$, violating the assumption of Lemma 6.3.
(3) $p \notin [b_1, b_4]$: If so then, by part (3) of Claim 6.8, we would have $\pi(p) \in P_{N-1}(\zeta)$, again violating the assumption of Lemma 6.3.
(4) π restricts to an injection of $P_N(z)$ into $P_N(\zeta)$: π is obviously an injection if restricted to the complement of $[b_1, b_4]$.
(5) π is not a surjection of $P_N(z)$ onto $P_N(\zeta)$: The only positions in z which might map to the middle position m in ζ are in $[c_1, a_4] \subseteq [b_1, b_4]$.

But then statements (4) and (5) imply that $P_N(z)$ has smaller cardinality than $P_N(\zeta)$, and we have finished the proof of Lemma 6.3, and hence of Theorem 6.1. □

Observe that if $L(Q, R) \backslash L_k(Q, R)$ is finite for some k then $L(Q, R) = L_N(Q, R)$ for some N, because each element of $L(Q, R)$, and hence each element of $L(Q, R) \backslash L_k(Q, R)$, is in some $L_j(Q, R)$. This observation allows us to reformulate the convergence theorem as a dichotomy:

Corollary 6.9. *With the terminology of Theorem* 6.1, *one of the following must hold*:

(1) $L_N(Q, R) = L(Q, R)$ *for all* $N \geqslant 2n$, *or*
(2) $L(Q, R) \backslash L_N(Q, R)$ *is infinite for all* N.

Now here is our main application:

Theorem 6.10. *Suppose* $t \in \{r, rs\}$. *Suppose* L *is a regular language, let* $n = \mathcal{N}(L)$, *and let* σ_{2n} *be the splicing scheme consisting of all rules of* $R^t(L)$ *of length at most* $2n$. *Then* L *is in the class* H^t *if and only if* $L \backslash \sigma_{2n}(L)$ *is finite.*

Proof. Set $R = R^t(L)$ and $Q = Q(L)$; clearly $\bar{R} = R$. Then Theorems 5.2 and 6.1 prove the result with $n = \mathcal{N}(Q, R) = \mathcal{N}(Q(L), R^t(L))$. Then we apply Lemma 4.2 and Theorems 4.9 and 3.3(3) to replace $\mathcal{N}(Q(L), R^t(L))$ with $\mathcal{N}(L)$. □

If a regular language L is specified constructively (for example, as the language accepted by a given finite automaton) then Syn L can be algorithmically constructed. Since the various constructions required by Theorem 6.10 only involve regular languages and can be performed by well-known algorithms, we have a decision theorem as a corollary:

Corollary 6.11. *Suppose* $t \in \{r, rs\}$. *There is an algorithm which determines whether a given regular language* L *is in the class* H^t. *In case the language is in* H^t *the algorithm constructs a finite set* L_0 *and a finite* H *scheme with rules in* $R^t(L)$ *so that* $L = \sigma^*(L_0)$.

Remark 6.12. It remains an open question to provide such an algorithm if the reflexivity assumption is dropped.

7. Constants

The notion of a *constant* was first defined by Schützenberger [18]: a word c is a *constant of the language* L if it satisfies the following condition: for any strings x_1, y_1, x_2, and y_2, if $x_1 c y_1$ and $y_2 c x_2$ are in L then $x_1 c x_2$ is in L. We write Const L for the set of all constants of L.

Notice the similarity between the statement that c is a constant of L and the statement that $r = \langle u, v, u, v \rangle$ respects L: this means that if $x_1 u v y_1$ and $y_2 u v x_2$ are in L then $x_1 u v x_2$ is in L. Exploiting this connection between constants and splicing, we can immediately specialize the results of Section 4 as follows.

Lemma 7.1. *For any language L:*

(1) *If $r = \langle u, v, u, v\rangle$ then r respects L iff uv is a constant of L.*
(2) *A rule is in $R^r(L)$ iff its sites are constants of L.*
(3) *L syntactically respects* Const L, *so* Const L *is regular if L is regular.*

A language L is said to be finitely based on constants (FBC) if there is a finite set of constants F of L so that all but finitely many of the words in L have a factor in F. The main motivation for this paper was the following theorem:

Theorem 7.2 (*Head [10]*). *Let $L \subseteq A^*$ be a regular language. Then the following are equivalent*:

(1) *$L = \sigma^*(L_0)$ where L_0 is finite and σ is a finite reflexive* H *scheme in which each rule is either of the form $\langle u, 1, v, 1\rangle$ or of the form $\langle 1, v, 1, u\rangle$.*
(2) *L is FBC.*

We may further require that the H *scheme in part* (1) *be symmetric.*

Symmetry was not in Head's original version, but it is obvious from his proof. The main innovation in Head's proof was the argument that FBC languages are splicing languages, and we have incorporated his idea in our Theorem 5.2, so it is not surprising that we have a short proof:

Proof. Part (1) implies part (2): There are finitely many sites of rules of σ, each is constant, and each result of a splicing operation contains a site. Since all but finitely many elements of L are the results of splicing, L is FBC.

Part (2) implies part (1): Let F be a finite set of constants of L so that $L\backslash A^*FA^*$ is finite and let σ_0 be the H scheme in which the rules are all tuples $\langle c, 1, c, 1\rangle$ or $\langle 1, c, 1, c\rangle$ where $c \in F$. Then $\sigma_0(L) \subseteq L$ and each word of $L \cap A^*FA^*$ is the result of splicing with itself using one of the rules of σ_0, so $L\backslash\sigma_0(L) \subseteq L\backslash A^*FA^*$. So Theorem 5.2 provides the desired H scheme σ. □

Example 8.1 provides a reflexive–symmetric splicing language which is not FBC.

Our algorithm for detecting reflexive splicing languages was motivated by Head's request in [10] for an algorithm to decide whether a given regular language is FBC. We answer this here as a consequence of Theorem 6.1. This answer, with a different proof, was first obtained by the first author in her dissertation [8].

Theorem 7.3. *Suppose L is a regular language and let $n = \mathcal{N}(L)$ as provided by the SPL. Let L_N be the set of words in L that contain a constant of L of length less than or equal to N. Then L is FBC if and only if $L\backslash L_{2n}$ is finite.*

Proof. Let $Q = \{q \in (A^*)^4 : q_1q_4 \in L\}$ and define a set of "rules" based on constants, $R = \{r \in (A^*)^4 : r_1 \in \text{Const } L\}$. We do *not* treat the elements of R as splicing rules, but simply as a technical device to help account for the presence of constants in words of L.

We first show that L_N equals $L_N(Q, R)$ as defined in Definition 4.5. If $q \in Q$, $r \in R$, $r \preccurlyeq q$, and $|r| \leqslant N$ then $q_1 = q_1'r_1$ for some q_1', so $\kappa(q) = q_1'r_1q_4$. Hence $\kappa(q) \in L$ and r_1 is a constant factor of w of length at most N. That is, $L_N(Q, R) \subseteq L_N$. Now suppose $w \in L_N$. Then we can factor $w = ucv$ with $c \in \text{Const } L$ and $|c| \leqslant N$. Define $q = \langle uc, 1, 1, v\rangle$ and $r = \langle c, 1, 1, 1\rangle$; we have $q \in Q$, $r \in R$, $r \preccurlyeq q$, and $\kappa(q) = w$. This provides the reverse inclusion, so $L_N = L_N(Q, R)$.

Let $L_* = L \cap \text{Const } L$, so L_* contains all words of L that contain a constant of L. Then the same argument as above shows that $L_* = L(Q, R)$, as defined in Definition 4.5.

Now L syntactically respects Const L by Lemma 7.1(3) and so L syntactically respects R. As in Lemma 4.2, L syntactically respects Q. Thus, Q and R are regular, and $\mathcal{N}(Q, R) \leqslant \mathcal{N}(L)$ follows from Theorem 3.3(3). Notice that any string which contains a constant of L is a constant of L, so $\bar{R} = R$.

Hence Corollary 6.9 applies, so either $L_{2n} = L_*$ or $L_*\backslash L_N$ is infinite for all N. Since $L \supset L_* \supset L_N$ the theorem follows. □

Corollary 7.4. *There is an algorithm which determines whether a given regular language L is FBC, and if so the algorithm constructs a finite set F of constants of L so that $L\backslash A^*FA^*$ is finite.*

Remark 7.5. It is possible to reduce $2n$ in Theorem 7.3 to n by working through the proof of Theorem 6.1 with this application in mind, or by reading the original proof in [8].

Remark 7.6. We do not know whether every splicing language must contain a constant. If this is the case then it should be very helpful in understanding the structure of general splicing languages.

8. Examples

We collect here a number of examples. We provide splicing languages that are reflexive–symmetric but not FBC, that are reflexive but not symmetric, symmetric but not reflexive, and neither reflexive nor symmetric. We also provide a regular language that is not a splicing language but does satisfy the condition of Theorem 5.2 (without the reflexivity requirement).

Example 8.1. $L = a^*b^*a^*$ is a reflexive–symmetric splicing language but L is not FBC.

Proof. Let σ be the reflexive–symmetric H scheme with rules $\langle b, 1, b, 1\rangle$, $\langle 1, b, 1, b\rangle$, $\langle 1, b, b, 1\rangle$, $\langle b, 1, 1, b\rangle$. Then $\sigma(L) = L$. Hence L is a reflexive–symmetric splicing language by Theorem 5.2. However, $L \supset a^*$ and no word in a^* can be a constant, so L is not FBC. □

The previous example fails to be FBC by having infinitely many non-constant words. The next fails by having infinitely many prime constant words. (A constant is *prime* iff it does not have a proper factor that is a constant.)

Example 8.2. $L = a^*ca^+b + ba^+ca^* + a^*ca^*ca^*$ is a reflexive splicing language but each word in $ca^*c \subset L$ is a prime constant of L so L is not FBC.

Proof. If σ is the reflexive H scheme with rules $\langle 1, ab, ba, 1\rangle$, $\langle 1, ab, 1, ab\rangle$, and $\langle ba, 1, ba, 1\rangle$ then $\sigma(L) = L$. Hence, by Theorem 5.2, L is a reflexive splicing language. Clearly every string of the form ca^kc is a constant. On the other hand, no string of the form a^j or a^jc or ca^j can be a constant, since any such constant could be used with elements in ba^+ca^* and a^*ca^+b to produce a string in bA^*b. Hence ca^kc is a prime constant. □

Example 8.3. $L = (aa)^*b + b(aa)^* + (aa)^*$ is a reflexive splicing language but is not a symmetric splicing language.

Proof. If σ is the reflexive H scheme with rules $\langle 1, ab, ba, 1\rangle$, $\langle 1, ab, 1, ab\rangle$, and $\langle ba, 1, ba, 1\rangle$ then $\sigma(L) = L\backslash\{1, b\}$. Hence, by Theorem 5.2, L is a reflexive splicing language.

Next, notice that no splicing rule r which respects L can have either r_1r_2 or r_3r_4 in a^*, since any such rule could be used to generate a word with an odd number of a's.

Now suppose L is symmetric, so $L = \sigma(L_0)$ where σ is a finite symmetric splicing scheme and L_0 is finite. Choose n large enough that $(aa)^n \notin L_0$. Then $(aa)^n$ is obtained from two strings of L by splicing with some rule r of σ, and by the discussion above we have $r = \langle a^i, a^jb, ba^k, a^m\rangle$. But then $\langle ba^k, a^m, a^i, a^jb\rangle$ applied to suitable words in $b(aa)^+$ and $(aa)^+b$ produces $ba^{k+j}b$, which is not in L, a contradiction. □

Example 8.4. Let $L = a^*ba^*ba^* + a^*ba^* + a^*$. Then L is a splicing language but neither a reflexive splicing language nor a symmetric splicing language.

Proof. We are using the alphabet $A = \{a, b\}$. Using the standard notation $|w|_b$ for the number of times b occurs in the string w, we can write $L = \{w \in A^*: |w|_b \leqslant 2\}$.

First we analyze the relevant rules for this language: we say a splicing rule r is *useful* for a language L if there are two words in L that can be spliced using r.

Claim 8.5. *For any* $r \in (A^*)^4$:

(1) *r is useful for L iff* $|r_1r_2|_b \leqslant 2$ *and* $|r_3r_4|_b \leqslant 2$.
(2) *If r is useful for L then r is in* $R(L)$ *iff* $|r_2r_3|_b \geqslant 2 \geqslant |r_1r_4|_b$.
(3) *If r is useful for L then r is in* $R^r(L)$ *iff* $|r_1r_2|_b = |r_3r_4|_b = 2$.
(4) *If r is useful for L and r is in* $R^s(L)$ *then r is in* $R^r(L)$.

Proof. Part (1): Obvious.

Part (2): Suppose r is useful and let $n_1 = 2 - |r_1r_2|_b$ and $n_2 = 2 - |r_3r_4|_b$. Then $w_1 = b^{n_1}r_1r_2$ and $w_2 = r_3r_4b^{n_2}$ are in L and splice, using r, to produce $b^{n_1}r_1r_4b^{n_2}$. If r respects L then

$$2 \geqslant |b^{n_1}r_1r_4b^{n_2}|_b = n_1 + |r_1r_2|_b - |r_2|_b + n_2 + |r_3r_4|_b - |r_3|_b = 2 + 2 - |r_2r_3|_b,$$

from which $|r_2r_3|_b \geqslant 2$ follows. The second inequality follows since $|r_1r_4|_b = |r_1r_2|_b + |r_3r_4|_b - |r_2r_3|_b \leqslant 2 + 2 - 2 = 2$.

Conversely, suppose $|r_2r_3|_b \geqslant 2$. Suppose $r \preccurlyeq q \in Q(L)$ and $\kappa(q) = q_1q_4 = z$. Then $|q_1|_b = |q_1r_2|_b - |r_2|_b \leqslant |q_1q_2|_b - |r_2|_b \leqslant 2 - |r_2|_b$, and similarly $|q_4|_b \leqslant 2 - |r_3|_b$. Then

$$|\kappa(q)|_b = |q_1|_b + |q_4|_b \leqslant 2 - |r_2|_b + 2 - |r_3|_b = 4 - |r_2r_3|_b \leqslant 2$$

and so $\kappa(q) \in L$. Hence $r(L) \subseteq L$.

Part (3): It is easy to check that a factor c of a word of L is a constant if and only if $|c|_b = 2$. Hence, a useful rule r is in $R^r(L)$ if and only if $|r_1r_2|_b = |r_3r_4|_b = 2$.

Part (4): Suppose a rule r is useful and is in $R^s(L)$. Part (2) applied to r gives $|r_1r_4|_b \leqslant 2 \leqslant |r_2r_3|_b$. The same inequalities hold for the reflection $\langle r_3, r_4, r_1, r_4\rangle$ of r, so $|r_3r_2|_b \leqslant 2 \leqslant |r_4r_1|_b$. Combining these inequalities proves $|r_1r_4|_b = |r_2r_3|_b = 2$. But from this we conclude $|r_1r_2|_b + |r_3r_4|_b = |r_1r_4|_b + |r_2r_3|_b = 4$. Since neither $|r_1r_2|_b$ nor $|r_3r_4|_b$ is greater than 2 we conclude $|r_1r_2|_b = |r_3r_4|_b = 2$. Then, by part (3), r is in $R^r(L)$. □

Now suppose L is a reflexive splicing language, so $L = \sigma^*(L_0)$ where L_0 is finite and σ is a finite H scheme with all rules in $R^r(L)$. We may assume that the rules of σ are useful. Let m be greater than the length of any word in L_0 and greater than twice the length of any rule in σ and consider the word $w = ba^mb$. Then $w \notin L_0$ so w is the result of splicing two words $x_1r_1r_2x_2$ and $x_3r_3r_4x_4$ of L using a rule r of σ. Hence $w = x_1r_1r_4x_2$. Since $|r_1r_2|b = 2 \geqslant |x_1r_1r_2x_2|_b$, x_1 cannot contain b, and similarly x_4 cannot contain b. Since w starts and ends with b we must have $x_1 = x_4 = 1$ so $w = ba^mb = r_1r_4$. This implies that $2|r| \geqslant |r_1r_4| > m$, contradicting the choice of m. Therefore L is not a reflexive splicing language.

By Claim 8.5(4), if $L = \sigma^*(L_0)$, where L_0 is finite and σ is a finite H scheme with all rules in $R^s(L)$, then all rules of σ would be in $R^r(L)$, which we have just seen is impossible. So L cannot be a symmetric splicing language.

Now we show that L is a splicing language. Let σ be the H scheme with rules

$$r^1 = \langle 1, 1, bb, 1\rangle, \quad r^2 = \langle 1, b, b, 1\rangle, \quad r^3 = \langle 1, bb, 1, 1\rangle$$

and let $L_0 = \{bb, bba, bab, abb\}$. By Claim 8.5(2) both $r^1(L) \subseteq L$ and $r^2(L) \subseteq L$ so $\sigma(L) \subseteq L$. We need to show that $L \subseteq \sigma^*(L_0)$. We do this in four phases.

First, generate bba^*: bb and bba are in L_0, and if we apply rule r^1 to bba^r and bba we produce bba^{r+1}.

Second, generate ba^*ba^*: bab is in L_0 and if we apply rule r^2 to bab and ba^qba^r we produce $ba^{q+1}ba^r$.

Third, generate $a^*ba^*ba^*$: abb is in L_0 and if we apply rule r^3 to abb and $a^pba^qba^r$ we produce $a^{p+1}ba^qba^r$.

Fourth, generate a^*ba^* and a^*: splicing a^pbb and bba^r using r^1 produces a^pba^r or a^{p+r} depending on which sites we use.

Hence L is a splicing language. □

Remark 8.6. It is not hard to extend the argument in Example 8.4 to show that the language $\{w \in \{a, b\}^* : |w|_b \leqslant N\}$ is a splicing language which is neither a reflexive splicing language nor a symmetric splicing language if $N \geqslant 2$.

We thank Fernando Guzmán for the following.

Example 8.7. $L = a^+b^+a^+b^+a^+ + a^+b^+a^+$ is a symmetric splicing language but is not a reflexive splicing language.

Proof. Let σ be the symmetric splicing scheme with rules

$$r^1 = \langle 1, abab, 1, aabab\rangle, \quad r^2 = \langle babaa, 1, baba, 1\rangle, \quad r^3 = \langle ba, b, b, a\rangle, \quad r^4 = \langle a, b, b, ab\rangle$$

and their "symmetric twins"

$$\bar{r}^1 = \langle 1, aabab, 1, abab\rangle, \quad \bar{r}^2 = \langle baba, 1, babaa, 1\rangle, \quad \bar{r}^3 = \langle b, a, ba, b\rangle, \quad \bar{r}^4 = \langle b, ab, a, b\rangle$$

and let $L_0 = \{a^2baba^2, aba\}$. We first show that $L = \sigma^*(L_0)$.

For this we require that $\sigma(L) \subseteq L$, which is left to the reader, and $L \subseteq \sigma^*(L_0)$. The latter occurs in four phases:

First, generate a^+baba^+: if $p \geqslant 2$ then apply r^1 to a^pbaba^2 and a^2baba^2 to produce $a^{p+1}baba^2$, and if $t \geqslant 2$ then apply r^2 to a^pbaba^2 and a^2baba^t to produce a^pbaba^{t+1}. Hence we can generate all of a^+baba^+ except strings a^pbaba^t with $p = 1$ or $t = 1$. But we can now derive these: if $p \geqslant 2$ and $t \geqslant 1$ we apply $\bar{r}^1$ to a^pbaba^2 and a^2baba^t to produce $a^{p-1}baba^t$, and similarly if $p \geqslant 1$ and $t \geqslant 2$ we apply $\bar{r}^2$ to a^pbaba^2 and a^2baba^t to produce a^pbaba^{t-1}.

Second, generate $a^+b^+ab^+a^+$: first, apply $\bar{r}^3$ to a^pbab^sa and $ababa^t$ to produce $a^pbab^{s+1}a^t$, and then apply $\bar{r}^4$ to a^pb^qaba and $abab^sa^t$ to produce $a^pb^{q+1}ab^sa^t$.

Third, generate $a^+b^+a^+b^+a^+$: apply r^3 to a^pb^qaba and $aba^rb^sa^t$ to produce $a^pb^qa^{r+1}b^sa^t$.

Finally, generate $a^+b^+a^+$: apply r^3 to a^pb^qaba and $ababa^t$ to produce $a^pb^qa^{t+1}$, or apply r^4 to a^pbaba and $abab^sa^t$ to produce $a^{p+1}b^sa^t$. This generates all of $a^+b^+a^+$ except aba, which is in L_0.

Now we check that L is not a reflexive splicing language, following the argument in Example 8.4. Suppose $L = \sigma^*(L_0)$ where L_0 is finite and σ is a finite H scheme with all rules in $R^r(L)$. Let m be greater than the length of any word in L_0 and greater than twice the length of any rule in σ and consider the word $aba^mba \in L\backslash L_0$. We note that the set of constant factors of L is $a^*b^+a^+b^+a^*$. Consider words $x_1r_1r_2x_2$ and $x_3r_3r_4x_4$ of L that splice, using a rule r of σ, to produce $x_1r_1r_4x_4 = aba^mba$. Since $x_1r_1r_2x_2$ is in L and r_1r_2 is in $a^*b^+a^+b^+a^*$ we conclude that x_1 is in a^+b^*, and similarly x_4 is in b^*a^+. Then r_1r_4 has a^m as a factor, which is impossible since $m > 2|r|$. □

Example 8.8. Let $L = a^* + ba^* + ba^*b$. Then there is a finite H scheme σ_0 so that $\sigma_0(L) = L$. However, L is not a splicing language.

Proof. Let σ_0 be the H scheme defined by the rules

$$r^1 = \langle ba, 1, ba, 1\rangle, \quad r^2 = \langle bb, 1, bb, 1\rangle, \quad r^3 = \langle 1, a, bb, 1\rangle.$$

Then $r^1(L) \subseteq L$ and $r^2(L) \subseteq L$ since ba and bb are constants of L. The only way to apply r^3 to elements of L is to splice xay and bb, producing x. That is, splicing with r^3 has the effect of removing any suffix which begins with a. Since L is closed under such operations we see that $r^3(L) \subseteq L$. Thus $\sigma_0(L) \subseteq L$.

For the opposite inclusion consider $w \in L$. If $w \in ba^+ + ba^+b$ then splicing w and w using r^1 produces w. If $w \in a^*$ then $w = a^j$ for some $j \geqslant 0$ and a^j is the result of splicing a^{j+1} and bb using r^3. These two cases cover all words of L except b, which is the result of splicing ba and bb using r^3, and bb, which is the result of splicing bb with itself using r^2. Thus $L \subseteq \sigma_0(L)$, and therefore $L = \sigma_0(L)$.

On the other hand, suppose L_0 is a finite subset of L and σ is a finite H scheme satisfying $\sigma(L) \subseteq L$. Let $m = 0$ if $L_0 \cap a^* = \emptyset$, and otherwise let m be the maximum integer n so that $a^n \in L_0$. We claim that $\sigma^*(L_0)$ cannot contain a^p for any $p > m$.

To prove the claim suppose that it is false. Then we can find k so that $\sigma^k(L_0)$ contains a^q where $q > m$ but $\sigma^{k-1}(L_0)$ does not contain any a^p with $p > m$. Since $a^q \notin \sigma^{k-1}(L_0)$ we can obtain a^q by splicing: there is a rule r of σ and there are words $w_1 = x_1r_1r_2x_2$ and $w_2 = x_3r_3r_4x_4$ in $\sigma^{k-1}(L_0)$ so that $a^q = x_1r_1r_4x_4$. We shall show this is impossible. There are two cases.

First, suppose $r_4x_4 = 1$. Then $x_1r_1 = a^q$ and $q > 0$, so w_1 begins with a. The only strings in L which begin with a are in a^* so $w_1 = a^n$. But $n \geqslant q$ since $x_1r_1 = a^q$ is a prefix of w_1, and this contradicts the choice of k.

Alternatively, suppose $r_4x_4 \neq 1$. Then w_2 is a string of L which ends in a so either $w_2 = a^n$ or $w_2 = ba^n$ for some n. Consider $\tilde{w}_2 = ba^nb$. This is in L and $\tilde{w}_2 = \tilde{x}_3r_3r_4x_4b$ where $\tilde{x}_3$ is either bx_3 (if $w_2 = a^n$) or x_3. Then w_1 and $\tilde{w}_2$ splice using r to produce $x_1r_1r_4x_4b = a^qb$. This contradicts the assumption that $\sigma(L) \subseteq L$.

Therefore, it is impossible to find L_0 and σ as we assumed, and so L cannot be a splicing language. □

Acknowledgements

Both authors would like to thank Tom Head for his support and inspiration and enthusiasm and encouragement during the first author's thesis research and also during the preparation of this paper.

References

[1] P. Bonizzoni, C. De Felice, G. Mauri, R. Zizza, Separating some splicing models, Inform. Process. Lett. 76 (2001) 255–259.
[2] P. Bonizzoni, C. De Felice, G. Mauri, R. Zizza, Regular languages generated by reflexive finite splicing systems, Lecture Notes in Computer Science, vol. 2710, 2003, pp. 134–145.
[3] P. Bonizzoni, C. De Felice, G. Mauri, R. Zizza, Decision problems for linear and circular splicing systems, Lecture Notes in Computer Science, vol. 2450, 2003, pp. 78–92.
[4] P. Bonizzoni, C. De Felice, R. Zizza, The structure of reflexive, regular splicing languages via Schützenberger constants, Theoret. Comput. Sci. 334 (2005) 71–98.
[5] K. Culik II, N-ary grammars and the descriptions of mappings of languages, Kybernetika 6 (1970) 99–117.
[6] K. Culik II, T. Harju, Splicing semigroups of dominoes and DNA, Discrete Appl. Math. 31 (1991) 261–277.
[7] R.W. Gatterdam, Splicing systems and regularity, Internat. J. Comput. Math. 31 (1989) 63–67.
[8] E. Goode, Constants and splicing systems, Ph.D. Thesis, Binghamton University, 1999.
[9] T. Head, Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors, Bull. Math. Biol. 49 (1987) 737–759.
[10] T. Head, Splicing languages generated with one sided context, in: G. Paŭn (Ed.), Computing with Bio-Molecules—Theory and Experiments, Springer, Singapore, 1998, pp. 269–282.
[11] T. Head, G. Păun, D. Pixton, Language theory and molecular genetics. Generative mechanisms suggested by DNA recombination, in: G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, vol. 2, Springer, Berlin, Heidelberg, New York, 1996, pp. 295–60, Handbook of Formal Languages, vols. 1–3.
[12] T. Head, D. Pixton, E. Goode, Splicing systems: regularity and below, in: DNA Computing, Eighth International Workshop on DNA Based Computers, DNA8, Sapporo, Japan, June 10–13, 2002, in: M. Hagiya, A. Ohuchi (Eds.), Revised Papers, Lecture Notes in Computer Science, vol. 2568, Springer, 2003, pp. 262–268.
[14] G. Păun, G. Rozenberg, A. Salomaa, DNA Computing: New Computing Paradigms, Springer, Berlin, 1998.
[15] J.E. Pin, Varieties of Formal Languages, Plenum Press, London, 1986.
[16] D. Pixton, Regularity of splicing languages, Discrete Appl. Math. 69 (1996) 99–122.
[17] D. Pixton, Splicing in abstract families of languages, Theoret. Comput. Sci. 234 (2000) 135–166.
[18] M.P. Schützenberger, Sur certaines operations de fermeture dans les langages rationnels, Sympos. Math. 15 (1975) 245–253.