

Regularity of Splicing Languages

Dennis Pixton*

Department of Mathematical Sciences
 State University of New York at Binghamton
 Binghamton, NY 13902-6000
 dennis@math.binghamton.edu

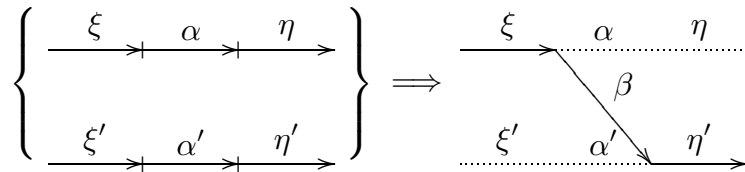
March 6, 1995

Abstract. Motivated by the recombinant behavior of DNA, Tom Head introduced a scheme for the evolution of formal languages called *splicing*. We give a simpler proof of the fundamental fact that the closure of a regular language under iterated splicing using a finite number of splicing rules is again regular. We then extend this result in two directions, by incorporating *circular strings* and by using infinite, but regular, sets of splicing rules.

SECTION 1. INTRODUCTION

In [3] and [4] Tom Head introduced an operation on strings called *splicing*. The basic idea is that two strings are cut at specified substrings, called *sites*, and the first segment of one is reattached to the second segment of the other with the sites suitably modified. The motivation for this operation lies in the study of the recombination of DNA fragments under the effects of restriction enzymes and ligases; we refer the reader to Head's papers for this motivation and for further references.

Our basic definition is a generalization of the definition presented in [4]. Throughout the paper we shall be working with strings over a fixed finite alphabet A . We say a triple of strings $r = (\alpha, \alpha'; \beta)$ is a *splicing rule*. Such a rule r may be applied to a pair of strings that contain the sites α and α' as substrings as follows. The first string is cut *before* α , the second is cut *after* α' , and the left segment of the first is attached to the right segment of the second with an intervening copy of β . Symbolically, if $\omega = \xi\alpha\eta$ and $\omega' = \xi'\alpha'\eta'$ then the effect of applying the rule r to these strings is the string $\xi\beta\eta'$. Graphically,



Of course ω and ω' may contain more than one copy of the sites α and α' , so the description of a splicing action must specify not just the original strings and the rule, but also the specific sites to be used. Head's original definition, more adapted to biochemical applications, corresponds to splicing rules of the special form $(\alpha\beta\gamma, \alpha'\beta\gamma'; \alpha\beta\gamma')$. A more recent version of the definition, used in [2] and [6], corresponds to splicing rules of the special form $(\alpha\beta, \alpha'\beta'; \alpha\beta')$.

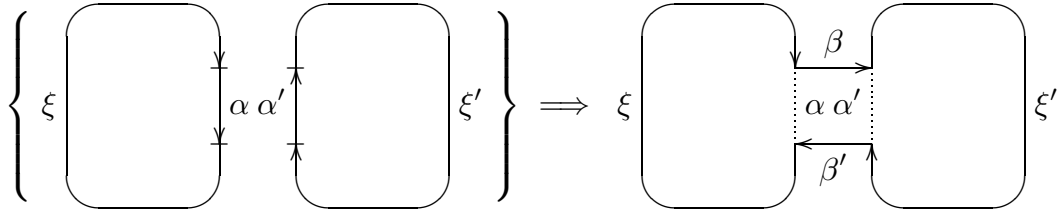
*Partially supported by NSF grant #CCR-9201345.

A finite collection R of splicing rules is called a *splicing scheme*. We are interested in the evolution of an initial set of strings L_0 under the actions of the rules in a splicing scheme. Precisely, we say a pair $\mathcal{S} = (R, L_0)$ is a *splicing system* if R is a splicing scheme and L_0 is a set of strings, called the *initial language*. We then define the *splicing language* determined by \mathcal{S} to be the smallest subset of A^* which contains L_0 and is invariant under splicing by rules in R . Equivalently, the splicing language is the set of strings which can be obtained from L_0 by repeated splicing using the rules in R .

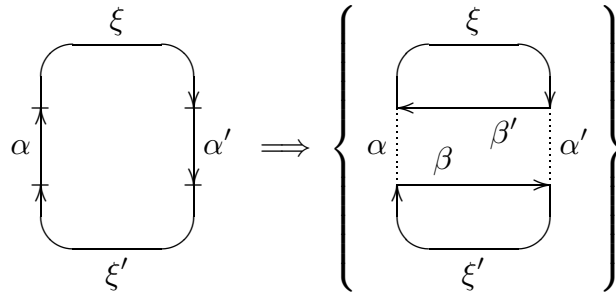
We address in this paper one of the fundamental questions posed in [3]: If the initial language is regular then is the splicing language regular? The answer is affirmative. Head gave a proof in his original paper, but with restrictions on the splicing scheme. The general result was proved by Culik and Harju in [1]. Their proof is somewhat complicated and is couched in the language of dominoes. Gatterdam [2] has presented a simpler proof, but still in a restricted setting. Our first result, presented in Section 2, is a considerable simplification of Culik and Harju's proof. This section also contains definitions and a basic construction which will be used in later sections.

However, the situation for *circular strings* is rather different. If ω is a string in A^* then $\hat{\omega}$ represents the circular form of ω , in which cyclic permutations of the letters in ω are identified. (Precise definitions and some basic properties are presented in Section 3.) The set of all such circular strings over A is denoted \hat{A} . These are natural objects of study in the biological context, since DNA occurs as loops as well as straight segments.

Clearly a single splicing operation on two circular strings will not produce a circular string. Instead, following [4], we assume that we have two splicing rules $r = (\alpha, \alpha'; \beta)$ and $r' = (\alpha', \alpha; \beta')$. Using this pair we can splice two circular strings $\hat{\omega} = \hat{\alpha\xi}$ and $\hat{\omega}' = \hat{\alpha'\xi'}$ containing the sites α and α' to obtain $\hat{\beta\xi'\beta'\xi}$:



There is another possibility. Suppose that $\hat{\omega} = \hat{\alpha\xi\alpha'\xi'}$ is a circular string containing both α and α' as disjoint subwords. Then we can cut this twice at the two sites and apply both splicing rules r and r' to obtain the two circular strings $\hat{\beta\xi'}$ and $\hat{\beta'\xi}$:



This action is called *self-splicing*. Note that the result is not simply the result of ordinary splicing performed on two separate copies of $\hat{\omega}$.

Just as in the linear case we can define a circular splicing system (R, C_0) where the initial language C_0 is a language in A^\wedge . Because of the requirement that splicing rules be applied in pairs it is reasonable to require that the splicing scheme R is *symmetric*; this just means that whenever a rule $(\alpha, \alpha'; \beta)$ is in R then there is another rule in R of the form $(\alpha', \alpha; \beta')$. The splicing language defined by such a system is then a language of circular strings. There is an obvious notion of regularity for circular languages (see Section 3), so we can again ask whether the circular splicing language defined by a symmetric splicing scheme with a regular initial language is regular.

Siromoney, Subramanian and Dare in [8] give an example of a circular splicing system with a finite initial language which determines a non-regular splicing language. Their example ignores self-splicing, but even if this is added the same example gives a non-regular splicing language. Their example, with variants, is presented in Section 4. On the positive side, they prove that the splicing language is regular, but only for a splicing scheme in which all rules have the form $(a, a; a)$ with $a \in A$.

Our primary new result is that the splicing language determined by a circular splicing system with a regular initial language is regular, provided that the splicing scheme R is symmetric and *reflexive*. This second assumption means that whenever α occurs as a site for some rule in R then the rule $(\alpha, \alpha; \alpha)$ lies in R . The intuition behind this assumption is just that the left and right ends of a cut at a site α should be allowed to reattach. The proof of our circular regularity theorem is in Section 5. It uses the same general method as in the linear case. Moreover, it uses critically the following consequence of the reflexivity assumption: If $\hat{\omega} \in C_0$ contains a site then $\hat{\omega}^n$ is in the splicing language for all positive integers n . This property is important since it is a necessary condition for a suitable representation of regular circular languages via automata; see Theorem 3.3.

Although reflexivity is a natural assumption it is not necessary for regularity in the circular case, although it is essential to our approach. A necessary and sufficient condition is unknown at this time.

We consider two further variations on the splicing theme, both of which are found in [4].

First, it is natural to consider mixtures of linear and circular languages. In this case there are two new ways that strings can be spliced: we can splice a linear string with a circular string to give a linear string, and we can self-splice a linear string to give a circular string. But this extra generality can be readily reduced to the purely circular case. See Section 6 for details.

Second, in the applications to biology it is important to remember that fragments of DNA are objects in three dimensions, so we should be able to reverse orientation on our strings. For biochemical reasons (the base pair duality) we must perform an involution on the letters of the alphabet when we reverse orientation. Thus we assume that we are given an involution $a \mapsto \tilde{a}$ on the alphabet, inducing orientation reversing involutions $\omega \mapsto \tilde{\omega}$ on A^* and $\hat{\omega} \mapsto \hat{\tilde{\omega}}$ on A^\wedge . Then we may require that the splicing operations do not distinguish between ω and $\tilde{\omega}$, or $\hat{\omega}$ and $\hat{\tilde{\omega}}$. This leads to two further splicing operations, linear and circular *inverted self-splicing*, in which we splice a string to itself, but considering it with two different orientations. Again these extra requirements can be readily subsumed into our earlier results. The details are in Section 7.

A generalization in a different direction was recently proposed by Gheorghe Păun in [6]. This concerns the possibility of moving beyond a finite set of rules, but maintaining some

degree of control by requiring that the rules, in an appropriate sense, form a regular set. Our analog of Păun’s definition and the corresponding regularity result are in Section 8. However, in the case of infinite splicing schemes our definition is *not* a generalization of Păun’s. Indeed, Păun has recently shown that, with his formulation, a regular splicing scheme with a finite initial language can produce a splicing language which is not even context sensitive. See [7].

ACKNOWLEDGMENT. I would like to thank Tom Head and Fernando Guzmán for suffering through some very complicated early versions of this work and for responding with several helpful suggestions.

SECTION 2. LINEAR REGULARITY

We shall use without comment the basic facts about regular languages; this is standard and can be found in most texts on formal languages. Our proofs of regularity use automaton methods. Since there are many variations on the basic idea of an automaton we record here our definitions. Recall that there is a fixed finite alphabet A . We write the empty word in A^* as 1. A *labeled directed graph* G is a directed graph with a labeling function λ from the set of edges into $A \cup \{1\}$. An *automaton* is a triple $\mathcal{A} = (G, I, T)$ where G is a labeled directed finite graph and I and T are sets of vertices of G , called the *initial* and *terminal* vertices respectively. If E is the set of edges of G then we regard λ as a homomorphism of E^* into A^* and we regard a path in G as a word in E^* . This is not quite satisfactory for the empty path 1, since a path must have starting and ending vertices, so, if necessary, we will refer to the “empty path at v ” to indicate the empty path which starts at the vertex v . A path p in G is an *accepting path* in \mathcal{A} iff it starts in I and ends in T . Thus the empty path is an accepting path iff $I \cap T \neq \emptyset$. Finally, the *language accepted by* \mathcal{A} is $\{\lambda(w) : w \text{ is an accepting path in } \mathcal{A}\}$.

It is well-known that a language is regular if and only if it is the language accepted by an automaton.

Our goal is to prove the following.

THEOREM 2.1. *If $\mathcal{S} = (R, L_0)$ is a splicing system with a regular initial language then the splicing language determined by \mathcal{S} is regular.*

This result is due to Culik and Harju in [1] and uses a basic construction which we have simplified. Our other main simplification is to consolidate several different nested inductions and to package them as a single induction using a non-linear notion of complexity.

We start the proof by constructing a graph B to capture the splicing scheme. For each splicing rule $r = (\alpha, \alpha'; \beta)$ in R we construct a linear graph $B(r)$, called the *bridge* associated to r , with just enough edges to carry the word β . We refer to the initial and terminal vertices of this linear subgraph as $i(r)$ and $t(r)$ respectively, and we write $b(r)$ for the path in $B(r)$ from $i(r)$ to $t(r)$, so $\lambda(b(r)) = \beta$. We construct these bridges to be pairwise disjoint, and we let B be their union.

Since L_0 is a regular language we can find an automaton $\mathcal{A}_0 = (G_0, I, T)$ which accepts L_0 . We may assume that G_0 is disjoint from B . We shall first construct a sequence of automata $\mathcal{A}_k = (G_k, I, T)$ accepting languages L_k . Note that the initial and terminal sets I and T do not depend on k , so we only have to show how to obtain G_{k+1} from G_k .

We consider each rule $r = (\alpha, \alpha'; \beta)$ for which both α and α' occur as substrings of elements of L_k . For each such rule we consider all accepting paths in \mathcal{A}_k . If pqt is an accepting path with q starting at v and $\lambda(q) = \alpha$ but there is no edge in G_k from v to $i(r)$ then we construct a new edge, labeled by 1, from v to $i(r)$. Similarly we construct a new edge from $t(r)$ to v' , labeled by 1, iff there is an accepting path $p'q't'$ with q' ending at v' and $\lambda(q') = \alpha'$ but there is no edge in G_k from $t(r)$ to v' . An edge of the first type is called an *initial marker of level $k + 1$* and an edge of the second type is called a *terminal marker of level $k + 1$* . Finally, if the bridge $B(r)$ does not lie in G_k then we say that $B(r)$ has level $k + 1$.

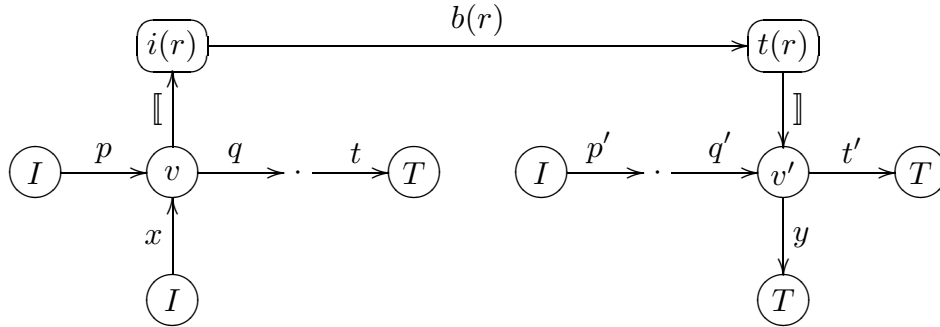
We obtain G_{k+1} from G_k by adding all markers and bridges of level $k + 1$.

It should be clear that initial and terminal markers play completely dual roles. This can be made formal by noticing that reversing direction on the strings of L_0 and on the graph G_0 and swapping I and T results in the interchange of the definitions for the two kinds of markers. We shall refer to this duality frequently.

NOTATION. We shall denote an initial or terminal marker by \llbracket or \rrbracket , respectively. If it becomes necessary to distinguish different markers we will use subscripts or accents.

The number of possible markers is finite (at most two times the number of components of B times the number of vertices in $G \cup B$) so eventually there is a level n for which $G_{n+1} = G_n$. To prove the theorem it is enough to show that the language L_n is closed under splicing and that L_n is a subset of the splicing language determined by \mathcal{S} .

To see closure under splicing, suppose that $\omega = \xi\alpha\eta$ and $\omega' = \xi'\alpha'\eta'$ are in L_n and that $r = (\alpha, \alpha'; \beta)$ is in R . Choose accepting paths pqt and $p'q't'$ such that $\lambda(p) = \xi$, $\lambda(q) = \alpha$, and $\lambda(t) = \eta$, and dually for p', q', t' . If q originates at v then there is an initial marker \llbracket from v to $i(r)$, and if q' terminates at v' then there is a terminal marker \rrbracket from $t(r)$ to v' . Hence the path $p\llbracket b(r)\rrbracket t'$ is an accepting path in \mathcal{A}_n and so $\xi\beta\eta' = \lambda(p\llbracket b(r)\rrbracket t')$ is in L_n .



To finish the proof we need to show that L_n is in the splicing language determined by \mathcal{S} . To do this we shall show how to “parse” any word $\lambda(w)$ where w is an accepting path in \mathcal{A}_n . That is, we shall show how to obtain $\lambda(w)$ by a finite number of splicings starting with elements of L_0 . This parsing will be controlled by the sequence of markers in w .

The proof depends on the following notion. We define the *complexity* of a path w in G_n as the n -tuple of natural numbers $c = \langle c_1, \dots, c_n \rangle \in \mathbb{N}^n$, where c_j is the number of markers in w of level j . We use a “right to left” lexicographic order on the set of

these tuples; that is, we write $c \prec d$ or $d \succ c$ iff there is k between 1 and n such that $c_j = d_j$ whenever $j > k$ but $c_k < d_k$. Since this is a total order (every non-empty set has a minimum element) we can use it as the basis for induction arguments.

We now argue, by induction on complexity, that any accepting path in \mathcal{A}_n determines a word in the splicing language.

First, an accepting path w of complexity $0 = \langle 0, \dots, 0 \rangle$ lies in G_0 , so $\lambda(w)$ is in L_0 .

So suppose that $c \in \mathbb{N}^n$ with $c \succ 0$ and suppose that for all accepting paths w' in \mathcal{A}_n of complexity $\prec c$ we have $\lambda(w')$ in the splicing language. Suppose that an accepting path w in \mathcal{A}_n has complexity c . Then w starts and ends in G_0 and it contains at least one marker, so it must visit at least one vertex in B . The only edges leading from G_0 to B are initial markers, so w contains at least one initial marker. Then we can select the last initial marker in w and write $w = x\llbracket u$ where u contains no initial markers. Since u connects a vertex of B to a terminal state in G_0 there must be a terminal marker in u . Selecting the first such terminal marker, we can write $w = x\llbracket z\rrbracket y$ where z contains no markers. We isolate the following simple observation for future reference.

LEMMA 2.2. *If $\llbracket z \rrbracket$ is a path in G_n and z contains no markers then there is a rule r such that \llbracket leads to $i(r)$, $z = b(r)$, and \rrbracket starts at $t(r)$.*

PROOF: \llbracket leads to some $i(r_1)$ and \rrbracket starts at some $t(r_2)$. Since z starts in $B(r_1)$ and cannot leave $B(r_1)$ but ends in $B(r_2)$ we must have $r_1 = r_2$ and $z = b(r_1)$. \square

Applying this to $w = x\llbracket z\rrbracket y$, let $r = (\alpha, \alpha'; \beta)$. Suppose \llbracket starts at v and has level k , and \rrbracket ends at v' and has level k' . Then there is an accepting path pqt in \mathcal{A}_{k-1} with q starting at v and $\lambda(q) = \alpha$, and there is an accepting path $p'q't'$ in $\mathcal{A}_{k'-1}$ with q' ending at v' and $\lambda(q') = \alpha'$. It follows that xqt and $p'q'y$ are accepting paths in \mathcal{A}_n and that $\lambda(xqt) = \lambda(x)\alpha\lambda(t)$ and $\lambda(p'q'y) = \lambda(p')\alpha'\lambda(y)$ splice together using the rule r to give $\lambda(x)\beta\lambda(y) = \lambda(w)$. We claim that xqt and $p'q'y$ have lower complexity than w . If we can establish this claim then, by the induction hypothesis, both $\lambda(xqt)$ and $\lambda(p'q'y)$ are in the splicing language, so $\lambda(w)$ is also in the splicing language, which finishes the proof.

So consider the complexity d of xqt ; the argument for $p'q'y$ is dual. Since qt is a path in G_{k-1} , all the markers in qt have level less than k . Thus if $j \geq k$ then any markers of level j in xqt must be in x and hence in w . This means that $d_j \leq c_j$ for all $j \geq k$. Moreover, xqt contains fewer markers of level k than $w = x\llbracket z\rrbracket y$ does (consider \llbracket), so $d_k < c_k$. Therefore $d \prec c$.

This concludes the proof of Theorem 2.1.

SECTION 3. CIRCULAR LANGUAGES

First some basic definitions.

Let \sim be the equivalence relation defined on A^* by cyclic permutation. That is, $\omega \sim \zeta$ iff $\omega = \xi\eta$ and $\zeta = \eta\xi$. The equivalence classes are called *circular strings*. The equivalence class of a string ω is written $\hat{\omega}$, and the set of all equivalence classes is written A^\wedge . We let $\pi: A^* \rightarrow A^\wedge$ be the quotient map sending ω to $\hat{\omega}$. If $L \subset A^*$ then we define $\text{Cir}(L) = \pi L$, the *circularization* of L . Conversely, if $C \subset A^\wedge$ then any $L \subset A^*$ for which $C = \text{Cir}(L)$ is called a *linearization* of C . Each $C \subset A^\wedge$ has a maximal linearization, $\text{Lin}(C)$, defined as $\pi^{-1}C$; we call this the *full linearization* of C .

Finally, we say that a circular language $C \subset A^\wedge$ is *regular* iff it has a regular linearization. In fact, we only need to check the full linearization:

PROPOSITION 3.1. $C \subset A^\wedge$ is regular if and only if $\text{Lin}(C)$ is regular.

PROOF: Suppose L is a regular linearization of C . Then $\hat{\omega} \in C$ if and only if $\hat{\omega} = \hat{\zeta}$ for some $\zeta \in L$. Hence $\text{Lin}(C) = \{\omega \in A^* : \exists \zeta \in L, \omega \sim \zeta\}$. In other words, $\text{Lin}(C) = \{\eta\xi : \xi \in L\}$. For the regularity of this set see Exercise 3.4c in [5, p.72]. \square

COROLLARY 3.2. The class of regular circular languages is closed under pairwise union, intersection, and difference.

PROOF: This follows immediately from the same fact in the linear case since $\text{Lin} = \pi^{-1}$ preserves set operations. \square

As an example, let $L = (a^2)^*b$. Then L is regular, so $C = \text{Cir}(L)$ is regular. Another linearization of C is $\{a^kba^k : k \in \mathbb{N}\}$, showing that it is not true that all linearizations of a regular circular language are regular. The full linearization is $\text{Lin}(C) = (a^2)^*\{b, aba\}(a^2)^*$.

We shall need an appropriate connection between regular circular languages and automata. Suppose that $\mathcal{A} = (G, I, T)$ is an automaton, and write E for the set of edges in G . We let $P \subset E^*$ be the set of closed paths in G . Then P is closed under cyclic permutation. The members of $\text{Cir}(P)$ are called *loops* in G . A non-empty loop \hat{x} is an *accepting loop* iff some (and hence any) representative x visits both I and T , and the empty loop $\hat{1}$ is an accepting loop iff $I \cap T \neq \emptyset$. The labeling function obviously extends to a map, still called λ , from $\text{Cir}(P)$ to A^\wedge . The *circular language accepted by \mathcal{A}* is $\{\lambda(x) : x \text{ is an accepting loop in } \mathcal{A}\}$.

It is not true, as in the linear case, that a circular language is regular if and only if it is accepted in this sense by an automaton. We need an extra condition. If L is a linear language then we say that L is *closed under repetition* iff $\omega^n \in L$ whenever $\omega \in L$ and $n > 0$. This definition also applies to circular languages since (as the reader may check) $(\hat{\omega})^n$ is well defined as $\hat{\omega}^n$. The following is the characterization we need.

THEOREM 3.3. A circular language $C \subset A^\wedge$ is the circular language accepted by an automaton if and only if C is regular and closed under repetition.

First we prove necessity. Let $\mathcal{A} = (G, I, T)$ be an automaton. If p and q are vertices in G then we let L_{pq} be the regular language accepted by the automaton $(G, \{p\}, \{q\})$. Then a circular path \hat{w} visits both I and T if and only if $\hat{w} = \hat{xy}$ where x starts at a vertex p in I and ends at a vertex q in T , and y starts at q and ends at p . Hence the circular language accepted by \mathcal{A} is $C = \text{Cir}\left(\bigcup_{pq} L_{pq}L_{qp}\right)$, with $p \in I$ and $q \in T$, so C is regular. If \hat{w} is an accepting loop then so is \hat{w}^n for all $n > 0$, so C is closed under repetition.

For the converse we shall use the following fact about linear languages.

LEMMA 3.4. A regular language L is closed under repetition if and only if there are finitely many regular languages L_1, \dots, L_m such that $L = \bigcup_k (L_k)^+$.

PROOF: Sufficiency is obvious. So we start with a regular language L which is closed under repetition. It is well known that we can find a *deterministic* automaton $\mathcal{A} = (G, I, T)$

that accepts L . *Deterministic* means that I is a singleton, $\{i\}$, and that for each vertex v in G and each word ξ in A^* there is a *unique* path x in G starting at v with $\lambda(x) = \xi$.

We now consider sequences $s = \langle s_0, s_1, \dots, s_n \rangle$ of vertices in G with the following properties:

- (a) $s_0 = i$.
- (b) $s_j \in T$ for all $j > 0$.
- (c) $s_n = s_{n-p}$ for some $p > 0$.
- (d) $s_i \neq s_j$ if $i \neq j$ and $i < n, j < n$.

Clearly there are finitely many such sequences s , and p is uniquely determined by s . We extend the sequence beyond s_n using periodicity of period p . That is, $s_j = s_{j-p}$ if $j > n$. For vertices v, w in G we let $L(v, w)$ be the language accepted by $(G, \{v\}, \{w\})$, and for each sequence s we define

$$L_s = L(s_0, s_1) \cap L(s_1, s_2) \cap \dots \cap L(s_{n-1}, s_n).$$

We shall finish the proof by showing that $L = \bigcup_s (L_s)^+$.

First, suppose $\xi_1, \dots, \xi_N \in L_s$. Then for each j there is a path x_j from s_{j-1} to s_j with $\lambda(x_j) = \xi_j$. Since $x = x_1 x_2 \dots x_N$ connects $s_0 = i$ to $s_N \in T$ it is an accepting path with $\lambda(x) = \xi_1 \dots \xi_N$. This shows that $(L_s)^+ \subset L$.

For the second inclusion, suppose $\xi \in L$. Then, for any $j > 0$, ξ^j is accepted by a path x_j leading to a vertex s_j of T . Since accepting paths are uniquely determined by their labels we see that x_{j-1} is a prefix of x_j , so $x_j = x_{j-1} y_j$ where y_j connects s_{j-1} to s_j and $\lambda(y_j) = \xi$. This demonstrates that $\xi \in L(s_{j-1}, s_j)$ for all j . Hence, if n is the first index for which $s_n = s_{n-p}$ for some $p > 0$, we have $\xi \in L_s \subset (L_s)^+$. \square

We can now complete the proof of Theorem 3.3. Suppose that $C \subset \hat{A}$ is a regular circular language that is closed under repetition. We start with the full linearization $L = \text{Lin}(C)$, which is also closed under repetition. Via Lemma 3.4 we can write $L = \bigcup_{k=1}^m (L_k)^+$ for regular languages L_k .

We shall construct an automaton $\mathcal{A} = (G, I, T)$ which accepts C . For each k let $\mathcal{A}_k = (G_k, I_k, T_k)$ be an automaton which accepts L_k . For each k we introduce a new vertex v_k in G_k , and we add edges labeled by 1 connecting v_k to each vertex in I_k and connecting each vertex in T_k to v_k . We denote the modified graph by \tilde{G}_k . We may arrange that the graphs \tilde{G}_k are pairwise disjoint. Now let G be the union of these graphs, and define $I = \{v_k : 1 \leq k \leq m\}$ and $T = \bigcup_k T_k$.

Since the graphs \tilde{G}_k are disjoint, a loop \hat{w} in G must lie in one \tilde{G}_k . It is then clear that \hat{w} is an accepting loop in \mathcal{A} if and only if it can be written in the form $\hat{w} = \hat{e}_1 x_1 f_1 e_2 x_2 f_2 \dots e_m x_m f_m$ where $m > 0$, e_j is an edge from v_k to a vertex in I_k , x_j is an accepting path in \mathcal{A}_k , and f_j is an edge from a vertex in T_k back to v_k . From this representation it follows immediately that the circular language accepted by \mathcal{A} consists of all circular words $\hat{\xi}_1 \hat{\xi}_2 \dots \hat{\xi}_m$ where the ξ 's all lie in the same L_k . That is, the circular language accepted by \mathcal{A} is $\bigcup_k \text{Cir}((L_k)^+) = \text{Cir}(\bigcup_k (L_k)^+) = \text{Cir}(L) = C$, as desired.

SECTION 4. CIRCULAR NON-REGULARITY

In this section we repeat the non-regularity example of Siromoney, Subramanian and Dare from [8], together with some variations. First the original example:

EXAMPLE 4.1. Let $A = \{a, b\}$, $R = \{(ab, ba; aa), (ba, ab; bb)\}$, and $C_0 = \{\hat{ab}\}$. Let C_1 be the language determined by this splicing system, **without using self-splicing**. Then $C_1 = \{\hat{a}^n b^n : n > 0\}$, which is not regular.

PROOF: The only splicing sites on $\hat{a}^n b^n = \hat{b}^n a^n$ are ab and ba . If $\hat{a}^n b^n$ is spliced to $\hat{b}^m a^m$ using these sites the result is $\hat{a}^{n+m} b^{n+m}$. This verifies closure under splicing, and also shows that $\hat{a}^n b^n$ may be obtained by repeated splicing with \hat{ab} , starting with \hat{ab} .

Non-regularity in this and the next two examples follows easily from a pumping argument applied to the full linearizations. \square

Adding self-splicing does not help:

EXAMPLE 4.2. If C_2 is the language determined by the splicing system of Example 4.1 then $C_2 = C_1 \cup \text{Cir}(a^2 a^* \cup b^2 b^*)$, which is not regular.

PROOF: \hat{ab} does not contain disjoint splicing sites. For $n \geq 2$ there are disjoint substrings ab and ba in $\hat{a}^n b^n$, and the results of self-splicing are \hat{a}^n and \hat{b}^n . Since these do not contain splicing sites they do not participate in further splices. \square

Thus something is needed beyond the basic definitions to ensure regularity. Reflexivity, without self-splicing, is not enough:

EXAMPLE 4.3. Let (R, C_0) be the splicing system of Example 4.1 and let $\bar{R} = R \cup \{(ab, ab; ab), (ba, ba; ba)\}$. Let C_3 be the language determined by the splicing system (\bar{R}, C_0) , **without using self-splicing**. Then $C_3 = \{\hat{\omega} \in A^+ : |\omega|_a = |\omega|_b > 0\}$, which is not regular. ($|\omega|_c$ is the number of occurrences of the letter c in ω .)

PROOF: Splicing $\hat{\omega}$ and $\hat{\omega}'$ according to $(ab, ba; aa)$ and $(ba, ab; bb)$ has the effect of deleting ab from $\hat{\omega}$ and ba from $\hat{\omega}'$ and joining the remnants together using aa and bb . Thus, if $\hat{\zeta}$ is the result, $|\zeta|_a = |\omega|_a + |\omega'|_a$, and similarly for the count of b 's. If the splicing is according to $(ab, ab; ab)$ (twice) then similar considerations again yield $|\zeta|_a = |\omega|_a + |\omega'|_a$. The other two splicing possibilities are symmetric in b and a . This proves closure under splicing.

We now show how to obtain any $\hat{\omega}$ with $|\omega|_a = |\omega|_b > 0$ by splicing. We proceed by induction on the length of ω . If this length is 2 then $\hat{\omega} = \hat{ab} \in C_0$. So assume that the length of ω is at least 4.

First assume that $\hat{\omega}$ contains the substring $a^2 b^2$, so $\hat{\omega} = \hat{a}^2 b^2 \zeta$. Then $\hat{\omega}$ is obtained by splicing $\hat{ab}\zeta$ and \hat{ba} , using $(ab, ba; aa)$ and $(ba, ab; bb)$ applied at the underlined sites. Hence, by induction, $\hat{\omega}$ is in C_3 .

So we may assume that $\hat{\omega}$ does not contain $a^2 b^2$ as a substring, and that ω has length at least 4. We claim that $\hat{\omega}$ must contain $abab$ as a substring. To see this, write $\hat{\omega} = \hat{a}^{j_1} b^{k_1} \dots \hat{a}^{j_r} b^{k_r}$ with $r > 0$, $j_i > 0$ and $k_i > 0$. If all $j_i = 1$ then $|\omega|_a = r$ so we must have all $k_i = 1$. Hence $\hat{\omega} = \hat{(ab)}^r = \hat{abab}(ab)^{r-2}$. Otherwise, we may assume that $j_1 > 1$. Note that $j_i > 1$ implies $k_i = 1$. If all $j_i > 1$ then all $k_i = 1$, which implies (as above) that all $j_i = 1$. So there is a first i for which $j_i = 1$. Then $k_{i-1} = 1$, so $\hat{a}^{j_{i-1}} b^{k_{i-1}} \hat{a}^{j_i} b^{k_i}$ contains $abab$.

Now we may write $\hat{\omega} = \hat{abab}\zeta$, which is obtained by splicing $\hat{ab}\zeta$ and \hat{ab} , using the rule $(ab, ab; ab)$ twice at the underlined sites. Again we conclude, by induction, that $\hat{\omega}$ is in C_3 . \square

Finally, if we incorporate both reflexivity and self-splicing we get regularity:

EXAMPLE 4.4. Let C_4 be the language determined by the splicing system of Example 4.3. Then $C_4 = \text{Cir}(\{a, b\}^2\{a, b\}^*)$, which is regular.

PROOF: Clearly, words of length less than 2 are not in C_4 . We need to show that any string $\hat{\omega}$ of length at least 2 is in C_4 . If ω contains only a 's or only b 's then $\hat{\omega}$ is in C_2 , and we are done. So we may suppose that $\hat{\omega}$ contains at least one a and at least one b , so it must contain ba , so it may be written as $\hat{a}\zeta b$. If $m = |\omega|_a$ and $n = |\omega|_b$ then $\hat{a}\zeta ba^n b^m$ is in C_3 . If we write this as $\hat{\zeta} \underline{ba} a^{m-1} b^{m-1} \underline{ba}$ then we see that self-splicing at the indicated sites using $(ba, ba; ba)$ twice yields $\hat{\zeta} ba = \hat{\omega}$, so $\hat{\omega}$ is in the splicing language. \square

SECTION 5. CIRCULAR REGULARITY

In this section we prove our main result, the analogue of Theorem 2.1 in the circular case.

THEOREM 5.1. Suppose that R is a symmetric and reflexive splicing scheme. If C_0 is a regular circular language then the circular splicing language determined by $\mathcal{S} = (R, C_0)$ is regular.

The proof uses the same type of construction as in Theorem 2.1, but with several modifications to accommodate circularity. The first step is to adjust C_0 so that it can be represented by an automaton.

Let S be the set of all sites used in the rules in R . For each $\alpha \in S$ we let $C_\alpha = C_0 \cap \text{Cir}(\alpha A^*)$; this is the set of strings in C_0 which contain the site α . Also, we define $L_\alpha = \text{Lin}(C_0) \cap \alpha A^*$; this is a regular linearization of C_α in which all strings begin with α . We claim that $\text{Cir}(L_\alpha^+)$ is in the splicing language C determined by \mathcal{S} . Specifically, if $\hat{\xi}_k = \hat{\alpha}\eta_k$ are in L_α for $k \leq n$ then $\hat{\omega}_n = \hat{\alpha}\eta_1\alpha\eta_2 \dots \alpha\eta_n$ is obtained by splicing $\hat{\xi}_n = \hat{\alpha}\eta_n$ and $\hat{\omega}_{n-1} = \hat{\alpha}\eta_1\alpha\eta_2 \dots \alpha\eta_{n-1}$ at the indicated sites using the rule $(\alpha, \alpha; \alpha)$ twice. We let $C'_0 = \bigcup_{\alpha \in S} \text{Cir}(L_\alpha^+)$. Then C'_0 is regular and closed under repetition, so it is the circular language accepted by some automaton. We let C' be the circular splicing language determined by (R, C'_0) . Since $C'_0 \subset C$ and the elements of $C_0 \setminus C'_0$ do not participate in any splicing operations (they don't contain any sites) we have $C = C' \cup (C_0 \setminus C'_0)$. Thus C is regular if C' is regular.

Hence, replacing C_0 by C'_0 , we may assume that C_0 is the circular language accepted by some automaton (G_0, I_0, T_0) . It is easy to arrange that $I_0 \cap T_0 = \emptyset$, by replacing any vertex v in $I_0 \cap T_0$ with a pair of new vertices $v_1 \in I_0$ and $v_2 \in T_0$ and adding edges labeled by 1 connecting v_1 to v_2 and v_2 to v_1 . Of course, any edge leading from v must be replaced by one starting at v_1 , and any edge leading to v must be replaced by one ending at v_2 . We adjust the automaton further by removing all edges which do not lie on accepting loops and all isolated vertices. Summarizing, our starting position is:

5.2. C_0 is the circular language accepted by an automaton $\mathcal{A}_0 = (G_0, I_0, T_0)$ in which I_0 and T_0 are disjoint, any edge lies on an accepting loop, and there are no isolated vertices.

We now describe the construction of an automaton which accepts the circular splicing language. This is a modification of the construction in Section 2.

We define the bridges $B(r)$, vertices $i(r)$ and $t(r)$, and paths $b(r)$ as before, with the following modification: If $r = (\alpha, \alpha'; 1)$ then we insist that $B(r)$ have one edge, labeled by 1. This has the effect that $i(r) \neq t(r)$ for any r .

The construction of the automaton \mathcal{A}_{k+1} from $\mathcal{A}_k = (G_k, I_k, T_k)$ differs from the earlier construction in three places.

First, and obviously, we must use loops rather than paths. That is, we only construct an initial marker from v to $i(r)$ if there is an accepting loop $\hat{p}q$ in G_k , where q starts at v and $\lambda(q) = \alpha$. Dually, we construct a terminal marker from $t(r)$ to v' only if there is an accepting loop $\hat{q}'p'$ in G_k , where q' ends at v' and $\lambda(q') = \alpha'$.

Second, we shall require the following:

5.3. *No initial marker starts at a bridge endpoint, and no terminal marker ends at such a vertex.*

Finally, the initial and terminal sets are not independent of k . Instead, I_k consists of the vertices of I_0 together with all vertices $i(r)$ which lie in G_k . T_k is defined similarly. From $i(r) \neq t(r)$, disjointness of the different bridges, and 5.2 we see that I_k and T_k are disjoint for all k .

As in the linear case we let n be the first integer for which $\mathcal{A}_{n+1} = \mathcal{A}_n$. We shall show that C_n , the circular language accepted by \mathcal{A}_n , is the splicing language determined by \mathcal{S} .

Before starting the proof we record some simple observations. We should note that part (b) below is the main reason we introduced the extra complication of 5.3. Also, it follows from part (d) that if $r = (\alpha, \alpha'; \beta)$ then $B(r)$ has level $k + 1$ iff the pair of substrings α and α' first appears in the language of \mathcal{A}_k , and so all other $B(r')$ where r' has the form $(\alpha, \alpha'; \beta')$ or $(\alpha', \alpha; \beta')$ will also have level $k + 1$.

LEMMA 5.4. *Suppose that \hat{w} is an accepting loop in \mathcal{A}_k which does not lie in G_0 . Then:*

- (a) \hat{w} visits $i(r_1)$ and $t(r_2)$ for rules r_1 and r_2 .
- (b) \hat{w} contains both an initial marker and a terminal marker.
- (c) \hat{w} does not contain a substring of any of the forms $\llbracket_1 \rrbracket_2$, $\llbracket_1 \rrbracket_2$ or $\llbracket_1 \rrbracket_2$.
- (d) If $\lambda(\hat{w}) = \hat{\xi}_1 \hat{\xi}_2 \dots \hat{\xi}_n$ then $\hat{w} = \hat{x}_1 x_2 \dots x_n$ where, for each i , $\lambda(x_i) = \xi_i$ and x_i does not start or end at a bridge endpoint.

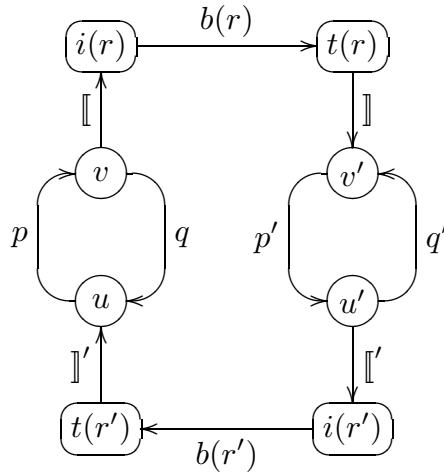
PROOF: Suppose that \hat{w} is an accepting loop and \hat{w} is not in G_0 . Then \hat{w} visits a vertex in some $B(r)$. If \hat{w} visits a vertex in G_0 then it must contain both an initial marker and a terminal marker, since these are the only ways to move from B to G_0 and back again. Hence \hat{w} visits vertices $i(r_1)$ and $t(r_2)$. But this is also true if \hat{w} never visits a vertex of G_0 , since \hat{w} must then visit $I_k \setminus I_0$ and $T_k \setminus T_0$.

Since $I_k \cap T_k = \emptyset$, \hat{w} cannot be the empty loop. Hence, since the only edges entering $i(r_1)$ are initial markers and the only edges leaving $t(r_2)$ are terminal markers, \hat{w} must contain one of each.

There cannot be a path in G_k of the form $\llbracket_1 \rrbracket_2$ since \llbracket_1 ends at $i(r_1)$ and \llbracket_2 cannot start at such a vertex. The same argument rules out $\llbracket_1 \rrbracket_2$. We cannot have $\llbracket_1 \rrbracket_2$ since $i(r_1) \neq t(r_2)$.

For part (d), take any factorization $\hat{w} = \hat{x}_1 x_2 \dots x_m$ with $\lambda(x_i) = \xi_i$. In the following we interpret x_{m+1} as x_1 . For i in order from 1 to m , if x_i consists only of markers

then we move the markers to x_{i+1} and replace x_i with an empty path. Next we remove, temporarily, any segments x_i which are empty. Now notice that x_i ends at a vertex of the form $i(r)$ if and only if it has the form $z\llbracket$, and x_i ends at a vertex of the form $t(r)$ if and only if x_{i+1} has the form $\rrbracket z$. From this and the dual observation at the other endpoint we see that the requirement of (d) is just that no x_i has either of the forms $\rrbracket z$ or $z\llbracket$. But if either of these occurs then the unwanted marker can be moved to the preceding or following x_j . These adjustments can be performed simultaneously, since \hat{w} does not contain a substring of the form $\llbracket\rrbracket_2$, and no further adjustments are necessary since there can be no repeated markers of the same type and because no x_i consists only of markers. Finally, we replace any empty segments x_i which were removed earlier in their proper locations. \square



We can now give the first half of the proof of Theorem 5.1, that C_n is closed under splicing. This follows just as in the linear case, except for the complications introduced by 5.3. So suppose that $\hat{\omega} = \hat{\alpha}\xi$ and $\hat{\omega}' = \hat{\alpha}'\xi'$ are in C_n and that $r = (\alpha, \alpha'; \beta)$ and $r' = (\alpha', \alpha; \beta')$ are splicing rules. By Lemma 5.4(d) we can find an accepting loop $\hat{w} = \hat{p}q$ with $\lambda(\hat{w}) = \hat{\omega}$ and $\lambda(q) = \alpha$, such that q does not start or end at a bridge endpoint. We can find a similar accepting loop $\hat{p}'q'$ for $\hat{\omega}'$ with $\lambda(q') = \alpha'$. Let v and u be the first and last vertices on q , and let u' and v' be the first and last vertices on q' . Then by the construction of G_n there are markers \llbracket from v to $i(r)$, \rrbracket from $t(r)$ to v' , \llbracket' from u' to $i(r')$, and \rrbracket' from $t(r')$ to u . Thus $\hat{z} = \hat{\llbracket}b(r)\rrbracket p' \llbracket' b(r') \rrbracket' p$ is an accepting loop in \mathcal{A}_n and $\lambda(\hat{z}) = \beta\xi'\beta'\xi$, which is the result of splicing $\hat{\omega}$ and $\hat{\omega}'$.

This, together with a similar argument for self-splicing, finishes the proof that C_n is closed under splicing.

The remaining part of the proof is the demonstration that the language C_n is in the splicing language determined by \mathcal{S} . In this argument we shall need to construct various accepting loops; that is the point of the following two lemmas.

LEMMA 5.5. *If e is an edge of level k then there is an accepting loop in \mathcal{A}_k containing e of the form $\hat{z} = \hat{\llbracket}b(r)\rrbracket p' \llbracket' b(r') \rrbracket' p$, and the only markers in \hat{z} which can be of level k are the four displayed markers. Moreover:*

- (a) If e is an initial marker leading to a bridge of level less than k then \llbracket and \llbracket' have levels less than k .
- (b) If e is a terminal marker leading from a bridge of level less than k then \llbracket and \llbracket' have levels less than k .

PROOF: First consider an initial marker \llbracket of level k from a vertex v to $i(r)$. We write $r = (\alpha, \alpha'; \beta)$ and let $B(r)$ have level j ; we must have $j \leq k$. Select a rule $r' = (\alpha', \alpha; \beta')$. Then $B(r')$ also has level j and there are accepting loops $\hat{p}q$ in \mathcal{A}_{k-1} and $\hat{p}'q'$ in \mathcal{A}_{j-1} and markers $\llbracket, \llbracket', \llbracket'$ such that

- (a) $\lambda(q) = \alpha$ and $\lambda(q') = \alpha'$.
- (b) q starts at v and ends at u , and q' starts at u' and ends at v' .
- (c) \llbracket' connects $t(r')$ to u and has level at most k .
- (d) \llbracket' connects u' to $i(r')$ and \llbracket connects $t(r)$ to v' , and both of these markers have level j .

Then $\hat{\llbracket b(r) \rrbracket p' \llbracket' b(r') \rrbracket' p}$ is an accepting loop in \mathcal{A}_k .

A dual argument works for terminal markers of level k .

If e is an edge on a bridge $B(r)$ of level k then we can follow $B(r)$ to a terminal marker of level k and then use the same procedure as above to return to the beginning of $B(r)$. \square

LEMMA 5.6. *If x is any path in G_k then there is a path x' such that $\hat{x}x'$ is an accepting loop in \mathcal{A}_k . Moreover:*

- (a) If x starts on a bridge of level j then x' contains a marker and the last marker on x' is an initial marker of level j .
- (b) If x ends on a bridge of level j then x' contains a marker and the first marker on x' is a terminal marker of level j .
- (c) If x does not start or end on a bridge of level k then all level k markers in x' lie in subpaths of the form $\llbracket z \rrbracket$ where z contains no markers.

PROOF: Suppose that $x = x_1x_2 \dots x_n$, and suppose that the lemma is true for each x_i , so there are paths x'_i such that each $\hat{x}_ix'_i$ is an accepting loop in G_k . If we set $x' = x'_kx'_{k-1} \dots x'_1$ then $\hat{x}x'$ is a closed path in G_k . We shall apply this construction with different choices for the segments x_i .

We shall proceed by induction on k . For the initial case, suppose that $k = 0$. If $x \neq 1$ we write $x = x_1x_2 \dots x_n$ where each x_i is an edge in G_0 . Since every edge in G_0 lies on an accepting loop we can find paths x'_i as desired. Since x_i is an edge, x'_i visits exactly the same vertices as $\hat{x}_ix'_i$, so each x'_i visits both I_0 and T_0 . Hence $\hat{x}x'$ is an accepting loop in G_0 . If x is the empty path starting at the vertex v then, since G_0 has no isolated vertices, we can find an edge e starting at v . We apply the above argument to e to find e' , and we let $x' = ee'$.

We now suppose that $k > 0$ and that the lemma is true for paths in G_{k-1} . Empty paths will be handled as in the base case, so we assume that x is not empty. We also may assume that x contains at least one edge of level k . In this case we write $x = x_1x_2 \dots x_n$ where the segments x_i are either paths in G_{k-1} or edges of level k . If x_i lies in G_{k-1}

then we find x'_i in G_{k-1} by the inductive hypothesis, and if x_i is an edge of level k then we let x'_i be the complement of x_i in the accepting loop \hat{z}_i provided by Lemma 5.5. Then $\hat{x}x'$ is an accepting loop since there is at least one x_i which is an edge of level k , and \hat{z}_i contains at least two of each type of marker, so x'_i contains both an initial marker and a terminal marker.

Now suppose that x does not start or end on a bridge of level k . If e is an edge in x which lies on a bridge of level k then e lies in a subpath of x of the form $\llbracket b(r) \rrbracket$. We then factor x as $x_1x_2 \dots x_n$ where each segment x_i is either a path in G_{k-1} or a path of the form $\llbracket b(r) \rrbracket$ consisting entirely of edges of level k or a marker of level k which does not meet a bridge of level k . We can apply the argument above to this factorization. If $x_i = \llbracket b(r) \rrbracket$ then the complement of this path in the loop provided by Lemma 5.5 has exactly two markers of level k , and these are in a substring of the form $\llbracket' b(r') \rrbracket'$. Similarly, if x_i is a single marker of level k then the only possible level k marker in x'_i lies in a substring of the same form. This implies part (c).

We now give the argument for part (a). Suppose that x starts on a bridge $B(r)$ of level j . Then there is an initial marker \llbracket of level j leading to $i(r)$ and a segment u of the bridge from $i(r)$ to the start of x . Then $y = \llbracket ux$ is a path in G_k so there is a path y' such that $\hat{y}y'$ is an accepting path in \mathcal{A}_k . We let $x' = y'\llbracket u$. This satisfies the requirements since u does not contain any markers.

The argument for part (b) is similar, and can be applied simultaneously with the argument for part (a). \square

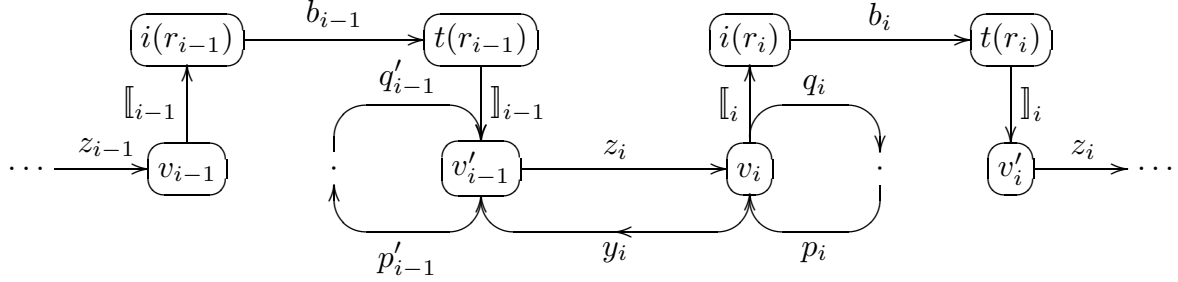
We are now prepared to show, by induction on k , that $\lambda(\hat{w})$ is in the splicing language defined by \mathcal{S} if \hat{w} is an accepting loop in \mathcal{A}_k . So we assume that $k > 0$ and that all accepting loops in \mathcal{A}_{k-1} determine words in the splicing language. We only need to consider loops which contain at least one level k marker.

We need a subsidiary induction based on a variant of the complexity used in Section 2. If $\llbracket' t \llbracket$ is a path and t contains no terminal markers then it is reasonable to expect that the level of \llbracket is strictly greater than the level of \llbracket' . This, however, is not necessarily so, and patterns where the level of \llbracket' is greater than or equal to the level of \llbracket cause difficulties. Suppose that \hat{w} is an accepting loop in \mathcal{A}_k . We shall say an initial marker \llbracket in \hat{w} is *k-inverted* iff it lies in a subsegment of \hat{w} of the form $\llbracket' t \llbracket$ where \llbracket' has level k and t contains no terminal markers. We have a dual definition for *k-inverted* terminal markers \rrbracket , considering segments of the form $\rrbracket t \rrbracket'$. We define the *complexity* of \hat{x} to be the k -tuple $c = \langle c_1, \dots, c_k \rangle$ where c_j is the number of level j markers in \hat{w} which are *k-inverted*. We use the reversed lexicographic order on complexities, as in Section 2.

We first consider an accepting loop \hat{w} of complexity 0. This means that there are no *k-inverted* markers at all. Also, \hat{w} contains at least one marker of level k and, by Lemma 5.4(b), it contains both an initial marker and a terminal marker. From this it follows easily that we can write $\hat{w} = \hat{z}_1 \llbracket_1 b_1 \rrbracket_1 \dots z_m \llbracket_m b_m \rrbracket_m$ where each pair $\llbracket_i, \rrbracket_i$ contains at least one level k marker, the segments b_i contain no markers, and the segments z_i lie in G_{k-1} . Here and in the discussion below we identify the subscripts 0 and m .

Using Lemma 2.2 we find rules $r_i = (\alpha_i, \alpha'_i; \beta_i)$ so that $b_i = b(r_i)$. We let the starting vertex of \llbracket_i be v_i and the ending vertex of \rrbracket_i be v'_i . We can find accepting loops $\hat{p}_i q_i$ and $\hat{p}'_i q'_i$ in \mathcal{A}_{k-1} such that q_i starts at v_i and $\lambda(q_i) = \alpha_i$, and q'_i ends at v'_i and $\lambda(q'_i) = \alpha'_i$.

Also, since z_i is a path in G_{k-1} connecting v'_{i-1} to v_i we can apply Lemma 5.6 to provide a path y_i in G_{k-1} leading from v_i to v'_{i-1} so that $\hat{z}_i y_i$ is an accepting path in \mathcal{A}_{k-1} . Finally we pick rules $r'_i = (\alpha'_i, \alpha_i; \beta'_i)$.

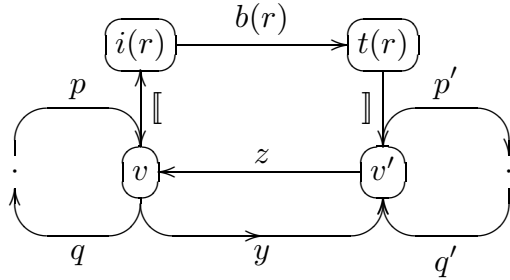


Now $\hat{x}_i = \hat{z}_i q_i p_i y_i p'_{i-1} q'_{i-1}$ is an accepting loop in \mathcal{A}_{k-1} so, by the main inductive hypothesis, $\hat{\xi}_i = \lambda(\hat{x}_i)$ is in the splicing language. Note that $\hat{\xi}_i = \zeta_i \alpha_i \eta_i \alpha'_{i-1}$ where $\zeta_i = \lambda(z_i)$ and $\eta_i = \lambda(p_i y_i p'_{i-1})$.

Consider $\hat{\xi}_1 = \hat{\zeta}_1 \alpha_1 \eta_1 \alpha'_0$ and $\hat{\xi}_2 = \hat{\zeta}_2 \alpha_2 \eta_2 \alpha'_1$. The result of splicing these at the underlined sites, using r_1 and r'_1 , is $\hat{\zeta}_1 \beta_1 \zeta_2 \alpha_2 \eta_2 \beta'_1 \eta_1 \alpha'_0$. It is then possible to splice this with ξ_3 , using α_2 and α'_2 as sites. The result, after $m-1$ such splices, is $\hat{\zeta}_1 \beta_1 \zeta_2 \beta_2 \dots \zeta_{m-1} \beta_{m-1} \zeta_m \alpha_m \eta_m \beta'_{m-1} \eta_{m-1} \beta'_{m-2} \dots \eta_2 \beta'_1 \eta_1 \alpha'_0$. Since $\alpha_m = \alpha_0$ we can apply self-splicing to σ at the underlined sites, using r_m and r'_m . The result of this operation is $\hat{\zeta}_1 \beta_1 \zeta_2 \beta_2 \dots \zeta_{m-1} \beta_{m-1} \zeta_m \beta_m = \lambda(\hat{w})$. Since the $\hat{\xi}_i$ are in the splicing language, we have shown that $\lambda(\hat{w})$ is in the splicing language.

To finish the proof of Theorem 5.1 we must provide the induction step based on complexity. So suppose that \hat{w} is an accepting loop in \mathcal{A}_k with complexity $c \succ 0$, and suppose that for any accepting loop \hat{w}' in \mathcal{A}_k with complexity $c' \prec c$ we have $\lambda(\hat{w}')$ in the splicing language.

Select a k -inverted marker in \hat{w} . Without loss of generality we may assume that this is an initial marker, \llbracket_0 . Let \llbracket be the first terminal marker appearing after \llbracket_0 and then let \llbracket be the last initial marker appearing before \llbracket . Since there are no terminal markers between \llbracket_0 and \llbracket , \llbracket is also k -inverted. From now on we concentrate on \llbracket and \llbracket . We can write $\hat{w} = \hat{\llbracket} b z$ where b contains no markers. By Lemma 2.2, $b = b(r)$ for some rule $r = (\alpha, \alpha'; \beta)$. We let j and j' be the levels of \llbracket and \llbracket respectively, and we choose a rule $r' = (\alpha', \alpha; \beta)$.



As in the base case (with $m = 1$) we find paths p, q, p', q' and y so that

- (a) $\hat{p}q$ is an accepting path in \mathcal{A}_{j-1} , q starts at the starting vertex v of \llbracket , and $\lambda(q) = \alpha$;

- (b) $\hat{p}'q'$ is an accepting path in $\mathcal{A}_{j'-1}$, q' ends at the end vertex v' of \llbracket , and $\lambda(q') = \alpha'$;
- (c) y is provided by Lemma 5.6 so that $\hat{z}y$ is an accepting path in \mathcal{A}_k .

Hence $\hat{w}' = \hat{q}pyp'q'z$ is an accepting path in \mathcal{A}_k and $\lambda(\hat{w}') = \hat{\alpha}\lambda(pyp')\hat{\alpha}'\lambda(z)$ can be self-spliced at the underlined sites, using r and r' , to yield $\hat{\beta}\lambda(z) = \lambda(\hat{\llbracket}b(r)\rrbracket z) = \lambda(\hat{w}')$. So we will be finished if we can show that the complexity c' of \hat{w}' is less than c .

It may happen that \llbracket is also k -inverted. In this case we may, without loss of generality, assume that $j \geq j'$. We shall show that $c' < c$ by analyzing the k -inverted markers in \hat{w}' . We shall show that $c'_i \leq c_i$ if $i > j$ and that $c'_j < c_j$. In fact, since \hat{w}' does not contain \llbracket , we only need to show that any k -inverted marker in \hat{w}' of level j or more is actually in z and remains k -inverted when considered in \hat{w} .

First we note that z starts and ends at the vertices v' and v , which are in G_{k-1} . Then z neither starts nor ends on a bridge of level k , so by Lemma 5.6(c) we may assume that all level k markers in y lie in subpaths of the form $\llbracket'x\rrbracket'$ where x contains no markers. Hence these markers cannot affect whether any other markers in \hat{w}' are k -inverted. Since p, q, p' and q' are in G_{k-1} , the only level k markers that affect k -inversion are in z .

Since \llbracket is k -inverted we can write $z = u\llbracket't$ where \llbracket' has level k and t contains no terminal markers. Hence z ends on a bridge, so by Lemma 5.6(b) we may assume the first marker in y is a terminal marker. Now suppose that \llbracket_0 is a k -inverted initial marker in \hat{w}' of level $i \geq j$. Then there is a substring of \hat{w}' of the form $\llbracket'_0 t_0 \llbracket_0$ where \llbracket'_0 has level k and t_0 has no terminal markers. By the preceding paragraph we must have \llbracket'_0 in z . Then t_0 must miss y since the first marker in y is a terminal marker, and \llbracket_0 does not lie in qp since qp is in G_{j-1} . Hence the path $\llbracket'_0 t_0 \llbracket_0$ lies entirely in z , so \llbracket_0 is k -inverted in \hat{w} .

We must also consider terminal markers \llbracket_0 of level at least j which are k -inverted in \hat{w}' . So suppose that there is a substring $\llbracket_0 t_0 \llbracket'_0$ of \hat{w}' with \llbracket'_0 of level k and with no initial markers in t_0 . Dualizing the argument above, we see that \llbracket'_0 must lie in z and, if \llbracket is itself k -inverted in \hat{w} , that $\llbracket_0 t_0 \llbracket'_0$ lies in z so \llbracket_0 is k -inverted in \hat{w} . In the remaining case \llbracket is not k -inverted in \hat{w} . That means we can write $z = t'\llbracket''u'$ where t' contains no terminal markers of level k . (We can find \llbracket'' since we know, from the preceding paragraph, that there is at least one initial marker, \llbracket' , in z .) Hence \llbracket'_0 must lie in u' , which then implies that $\llbracket_0 t_0 \llbracket'_0$ lies in u' , so, in this case as well, \llbracket_0 is k -inverted in \hat{w} .

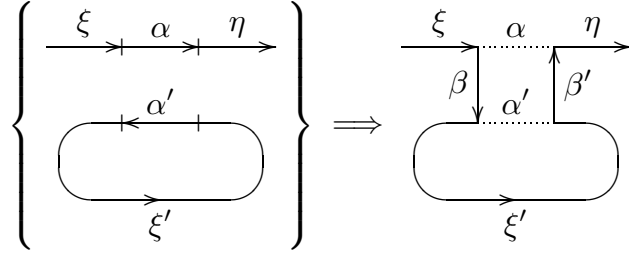
This finishes the induction on complexity and so finishes the proof of Theorem 5.1.

SECTION 6. MIXED SPLICING

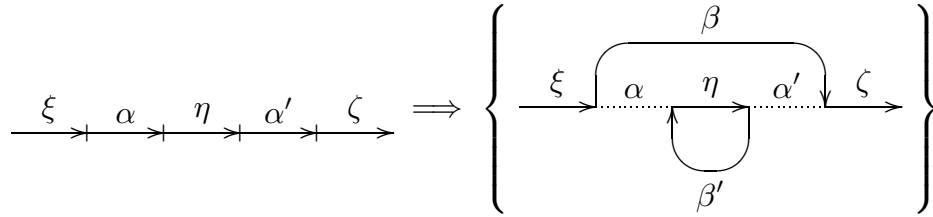
In [4] Head introduced not only circular splicing systems but also mixed splicing systems, in which the splicing rules apply to a mixture of linear and circular strings. Of course we need some extra splicing actions; otherwise the linear and circular subsets will develop independently. The extra actions allow a linear string and a circular string to splice to yield a linear string, and a linear string to self-splice to yield a circular string.

Suppose that $(\alpha, \alpha'; \beta)$ and $(\alpha', \alpha; \beta')$ are splicing rules. A linear string $\xi\alpha\eta$ containing α and a circular string $\hat{\alpha}'\xi'$ containing α' may be spliced to obtain the linear string

$\xi\beta\xi'\beta'\eta$:



Furthermore, a linear string $\omega = \xi\alpha\eta\alpha'\zeta$ containing α and α' as disjoint subwords can be self-spliced to obtain the linear string $\xi\beta\zeta$ and the circular string $\hat{\eta}\beta'$. Since $\xi\beta\zeta$ can be obtained by linearly splicing two copies of ω the only new string covered by this action is $\hat{\eta}\beta'$.



Thus we can talk about a *mixed splicing system* $\mathcal{S} = (R, M_0)$ where R is a splicing scheme and M_0 , the initial language, is a mixed language (that is, a subset of $A^* \cup \hat{A}$). The splicing language determined by \mathcal{S} is, as in the linear and circular cases, the smallest mixed language which contains M_0 and is closed under the linear, circular, and mixed splicing operations. As in the circular case we may as well assume that R is symmetric.

We say a mixed language M is regular iff both $M \cap A^*$ and $M \cap \hat{A}$ are regular. Siromoney, Subramanian and Dare presented an example in [8], similar to their example in Section 4, to show that the splicing language of a mixed splicing system is not necessarily regular, even if the initial language is finite. However, just as in the circular case, the imposition of reflexivity leads to regularity.

THEOREM 6.1. *Suppose that R is a symmetric and reflexive splicing scheme. If M_0 is a regular mixed language then the mixed splicing language determined by $\mathcal{S} = (R, M_0)$ is regular.*

PROOF: This result is a simple consequence of Theorem 5.1. To see this we add a new symbol, written \diamond , to the alphabet, and we associate to each linear string ω in M_0 the circular string $\hat{\omega}\diamond$. We let C_0 be the set of these associated circular strings together with the circular strings in M_0 . It is clear that C_0 is a regular circular language over the alphabet $A' = A \cup \{\diamond\}$. We let C be the circular splicing language determined by the circular splicing system (R, C_0) over A' .

Now we recapture the mixed splicing language determined by (R, M_0) as follows. If $\hat{\omega}$ is a circular string in C which does not contain \diamond then we put $\hat{\omega}$ in the set M . Otherwise if $\hat{\omega} = \hat{\xi}_1\diamond\xi_2\diamond\dots\xi_n\diamond$ with ξ_i in A^* then we put each ξ_i in M .

We leave it to the reader to check that M is indeed the splicing language determined by (R, M_0) and that regularity of C implies regularity of M . \square

SECTION 7. SPLICING WITH AN INVOLUTION

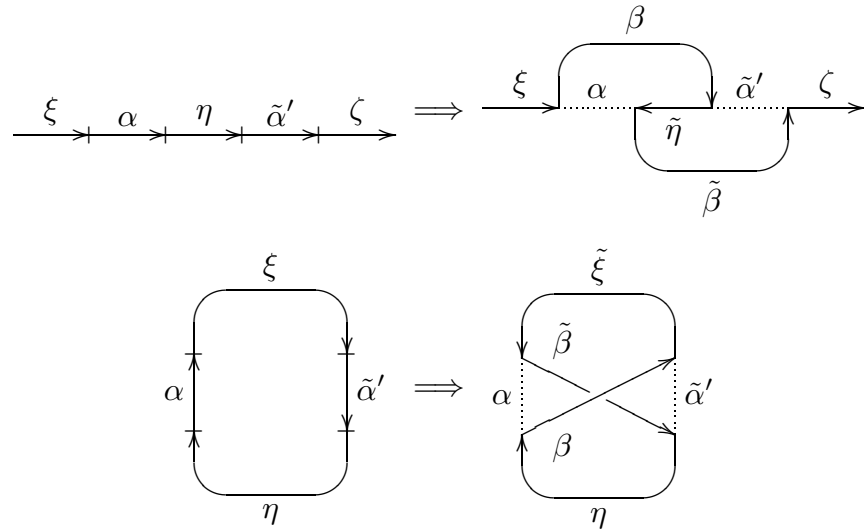
Following [4] we suppose we are given an involution, $a \mapsto \tilde{a}$, on the alphabet A . This extends to an involution $\omega \mapsto \tilde{\omega}$ on A^* by $a_1a_2 \dots a_m \mapsto \tilde{a}_m\tilde{a}_{m-1} \dots \tilde{a}_1$. This is also defined on A^\wedge . For the applications to DNA it is appropriate that strings be identified under the involution.

Rather than making this identification we just require that the initial language and the splicing scheme should be invariant under the involution. Here the requirement that a splicing scheme be invariant under the involution is that whenever $r = (\alpha, \alpha'; \beta)$ is in R then so is $\tilde{r} = (\tilde{\alpha}, \tilde{\alpha}'; \tilde{\beta})$. It is then a routine matter to verify that invariance is preserved, leading to:

THEOREM 7.1. *Suppose that R is a splicing scheme which is invariant under the involution. In the circular or mixed case assume that R is symmetric and reflexive. If the initial language in the linear, circular or mixed case is regular and invariant under the involution then so is the splicing language. \square*

However, there is another consideration. Since we want to identify strings under the involution we must consider the possibility that a string can splice with itself, but using two different orientations. That is, suppose $r = (\alpha, \alpha'; \beta)$ and suppose a string contains a copy of α and a later copy of $\tilde{\alpha}'$. Then we can follow the string in the original orientation to the beginning of α , jump via r to the *start* of $\tilde{\alpha}'$ (since this corresponds to the *end* of α' in the inverted string), follow the inverted string to the *end* of α (since this corresponds to the *start* of $\tilde{\alpha}$ in the inverted string), jump via \tilde{r} to the *end* of $\tilde{\alpha}'$, and continue in the original orientation along the string.

This operation is called *inverted self-splicing*. In the linear case it can be defined by $\xi\alpha\tilde{\alpha}'\zeta \implies \xi\beta\tilde{\eta}\tilde{\beta}\zeta$, and, in the circular case, by $\hat{\xi}\alpha\xi\tilde{\alpha}'\eta \implies \hat{\xi}\beta\tilde{\xi}\tilde{\beta}\eta$. Graphically,



In fact the effect of inverted self-splicing can be obtained just on the basis of the ordinary splicing operations. To see this, assume that we have already applied Theorem 7.1 to obtain a splicing language that is invariant under the involution.

In [4] Head gave the argument for the linear case. In fact, it is easy to check that the effect of linear self-splicing on ω can be obtained by two linear splices, one using r on ω and $\tilde{\omega}$, and the second using \tilde{r} on the result of this splice together with another copy of ω .

In the circular case we start with $\hat{\omega} = \hat{\underline{\alpha}}\xi\tilde{\alpha}'\eta$ and the rule $r = (\alpha, \alpha'; \beta)$. Since the splicing scheme is symmetric in the circular case we can find a rule $r' = (\alpha', \alpha; \beta')$. We use these rules to perform ordinary circular splicing on $\hat{\omega}$ and on $\hat{\tilde{\omega}} = \hat{\tilde{\eta}}\underline{\alpha}'\tilde{\xi}\tilde{\alpha}$ at the underlined sites. The result is $\hat{\beta}\xi\tilde{\alpha}\tilde{\eta}'\beta'\xi\tilde{\alpha}'\eta$. Now a self-splice operation on this string, using \tilde{r} and \tilde{r}' at the indicated sites, yields $\hat{\beta}\tilde{\xi}\tilde{\beta}\eta$, the result of inverted self-splicing.

Thus, in any case, inverted self-splicing is irrelevant to the construction of the splicing language.

SECTION 8. A REGULAR SET OF RULES

First we give the definition, analogous to that in [6], for a regular splicing scheme. We first add two new symbols, $\#$ and $\$$, to the alphabet to form an extended alphabet \bar{A} . A set R , possibly infinite, of splicing rules over the alphabet A is said to be a *regular splicing scheme* iff the set $L_R = \{ \alpha\#\alpha'\$\beta : (\alpha, \alpha'; \beta) \in R \}$ is a regular language in \bar{A}^* . We shall show that the splicing language determined by a regular initial language and a regular splicing scheme is regular. First we need a decomposition of the splicing scheme.

LEMMA 8.1. *A splicing scheme R is regular if and only if there are a positive integer m and regular subsets S_k, S'_k, T_k of A^* , for $1 \leq k \leq m$, such that $R = \bigcup_{k=1}^m S_k \times S'_k \times T_k$.*

PROOF: Let $\mathcal{A} = (G, I, F)$ be an automaton over \bar{A} which accepts L_R . For sets of vertices W and W' we let $L(W, W')$ be the language accepted by (G, W, W') . It is then clear that L_R is the union of the concatenations $L(I, \{v\})\#L(\{v'\}, \{w\})\$L(\{w'\}, F)$ under the conditions that there be an edge labeled by $\#$ connecting v to v' and an edge labeled by $\$$ connecting w to w' . We enumerate these concatenations as $S_k\#S'_k\$T_k$. Then $\alpha\#\alpha'\$\beta$ is in L_R if and only if $(\alpha, \alpha'; \beta)$ is in R , and $\alpha\#\alpha'\$\beta$ is in $S_k\#S'_k\$T_k$ if and only if $(\alpha, \alpha'; \beta)$ is in $S_k \times S'_k \times T_k$. \square

The following shows that we may assume that this decomposition is a finite partition of R .

LEMMA 8.2. *Suppose that m is a positive integer and that S_k, S'_k, T_k are subsets of A^* , for $1 \leq k \leq m$. Then there are a non-negative integer M and nonempty subsets $\bar{S}_j, \bar{S}'_j, \bar{T}_j$ of A^* , for $1 \leq j \leq M$, such that:*

- (a) *If all the S_k are regular then all the \bar{S}_j are regular, and similarly for S'_k and T_k .*
- (b) *The sets $\bar{S}_j \times \bar{S}'_j \times \bar{T}_j$ form a pairwise disjoint collection.*
- (c) $\bigcup_{k=1}^m S_k \times S'_k \times T_k = \bigcup_{j=1}^M \bar{S}_j \times \bar{S}'_j \times \bar{T}_j$.

PROOF: If \mathcal{S} is a finite collection of subsets of A^* then we let $\mathcal{R}(\mathcal{S})$ be the ring generated by \mathcal{S} . That is, $\mathcal{R}(\mathcal{S})$ is the smallest collection of subsets of A^* which contains \mathcal{S} and is closed under pairwise union, intersection and set difference. It is a standard exercise to show that $\mathcal{R}(\mathcal{S})$ is finite. We let $\mathcal{R}_0(\mathcal{S})$ be the collection of minimal (with respect to set inclusion) non-empty elements of $\mathcal{R}(\mathcal{S})$. This is a pairwise disjoint collection and each

element of $\mathcal{R}(\mathcal{S})$ is a union of elements of $\mathcal{R}_0(\mathcal{S})$. Since the collection of regular sets is closed under the basic set operations, if \mathcal{S} consists of regular sets then so does $\mathcal{R}(\mathcal{S})$.

We let \mathcal{S} , \mathcal{S}' and \mathcal{T} be the collections formed by the sets S_i , S'_i and T_i , respectively. Then $R = \bigcup_{k=1}^m S_k \times S'_k \times T_k$ is the disjoint union of the sets of the form $\bar{S}_j \times \bar{S}'_j \times \bar{T}_j$ which lie in R , where $\bar{S}_j \in \mathcal{R}_0(\mathcal{S})$, $\bar{S}'_j \in \mathcal{R}_0(\mathcal{S}')$ and $\bar{T}_j \in \mathcal{R}_0(\mathcal{T})$. \square

Hence the following implies that the splicing language determined by a regular splicing scheme and a regular initial language is regular, and is in fact a somewhat stronger theorem.

THEOREM 8.3. *Suppose that m is a positive integer and that S_ℓ , S'_ℓ , T_ℓ are subsets of A^* , for $1 \leq \ell \leq m$, with each T_ℓ regular, and define $R = \bigcup_{\ell=1}^m S_\ell \times S'_\ell \times T_\ell$. If L_0 is a regular subset of A^* then the splicing language determined by R and L_0 is regular.*

PROOF: By Lemma 8.2 we may assume that the sets $S_\ell \times S'_\ell \times T_\ell$ are pairwise disjoint. We require only a small modification to the basic construction in Section 2.

For each ℓ we let $\bar{\mathcal{B}}_\ell = (\bar{B}_\ell, I_\ell, F_\ell)$ be a deterministic automaton which accepts T_ℓ . We add a new vertex i_ℓ connected by edges labeled 1 to the vertices of I_ℓ and another new vertex t_ℓ connected by edges labeled 1 from the vertices of F_ℓ . We then remove all edges in \bar{B}_ℓ which are not on paths connecting i_ℓ to t_ℓ . The resulting automaton $\mathcal{B}_\ell = (B_\ell, \{i_\ell\}, \{t_\ell\})$ is no longer deterministic but it still accepts the language T_ℓ and it retains the property that any string in T_ℓ is the label of a unique accepting path. We may assume that the graphs B_ℓ are pairwise disjoint.

Now for each $r \in R$ we need a graph $B(r)$. We set $B(r) = B_\ell$ if $r \in S_\ell \times S'_\ell \times T_\ell$, and we set $i(r) = i_\ell$, $t(r) = t_\ell$. Also, if $r = (\alpha, \alpha'; \beta)$ then β is in T_ℓ so there is a unique accepting path $b(r)$ in \mathcal{B}_ℓ with $\lambda(b(r)) = \beta$. The rest of the construction of the automata \mathcal{A}_k proceeds as in Section 2.

The only differences between this construction and the original one are that the bridge graphs $B(r)$ are not linear graphs and that different rules may give rise to the same graph. If we examine the proof of Theorem 2.1 with these differences in mind we see that linearity of $B(r)$ is not used except to provide a path $b(r)$ with $\lambda(b(r)) = \beta$, and we have already provided this. The injectivity of the map $r \mapsto B(r)$ is used exactly once, in the proof of Lemma 2.2. So we need to reprove Lemma 2.2 in our current situation.

To do this, suppose that $\llbracket z \rrbracket$ is a path in G_k with z free of markers. Then \llbracket leads to $i(r_1)$ and \rrbracket starts at $t(r_2)$, with $r_i = (\alpha_i, \alpha'_i; \beta_i)$. The path z connects $i(r_1)$ to $t(r_2)$ and cannot leave $B(r_1)$ so $B(r_1) = B(r_2) = B_\ell$ for some ℓ . Hence both r_1 and r_2 are in $S_\ell \times S'_\ell \times T_\ell$ and z is an accepting path in \mathcal{B}_ℓ so $\beta = \lambda(z)$ is in T_ℓ . But then $r = (\alpha_1, \alpha'_2; \beta)$ is also in $S_\ell \times S'_\ell \times T_\ell$. Hence $i(r_1) = i(r) = i_\ell$ and $t(r_2) = t(r) = t_\ell$. Since accepting paths in \mathcal{B}_k are unique, $z = b(r)$. This is what is required for Lemma 2.2.

With these modifications, the proof of Theorem 2.1 also proves Theorem 8.3 \square

NOTE. The circular and mixed versions of this theorem also hold, including invariance under an involution, with the same extra hypotheses as in the case of a finite splicing scheme. There is one more piece of the original proof which uses finiteness of the splicing scheme and which must be modified. This is the proof of the preliminary reduction, 5.2, where the finite set of sites was used in constructing a replacement for the initial language which is closed under repetition. The modification to this construction involves replacing the finite set of sites with the finite set of factors S_ℓ and S'_ℓ , as above. We are then led to

consider sets of the form $C_0 \cap \text{Cir}(FA^*)$ where F is one of these factors. Since we want such sets to be regular we need the factors F to be regular, and this is true under the assumption of regularity of the splicing scheme, but not under the weaker assumptions of Theorem 8.3.

BIBLIOGRAPHY

- 1 Karel Culik II and Tero Harju, *Splicing semigroups of dominoes and DNA*, Discrete Appl. Math. **31** (1991), 261–277.
- 2 R. W. Gatterdam, *DNA and twist free splicing systems*, Words, Languages and Combinatorics II (M. Ito and H. Jurgensen, eds.), World Scientific, Singapore, to appear.
- 3 Tom Head, *Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors*, Bull. Math. Biol. **49** (1987), 737–759.
- 4 ———, *Splicing schemes and DNA*, Lindenmayer Systems – Impacts on Theoretical Computer Science, Computer Graphics, and Developmental Biology (G. Rozenberg and A. Salomaa, eds.), Springer Verlag, Berlin, 1992, pp. 371–383.
- 5 John Hopcroft and Jeffrey Ullman, *Introduction to automata theory, languages, and computation*, Addison-Wesley, 1979.
- 6 Gheorghe Păun, *On the splicing operation*, Discrete Appl. Math., to appear.
- 7 ———, *Regular extended H systems are computationally universal*, (1995), preprint.
- 8 Rani Siromoney, K. G. Subramanian, and V. Rajkumar Dare, *Circular DNA and splicing systems*, Parallel Image Analysis, Lecture Notes in Computer Science, vol. 654, Springer Verlag, Berlin, 1992, pp. 260–273.