

Semi-simple Splicing Systems

Elizabeth Goode*

CIS, University of Delaware
Newark, DE 19706
goode@mail.eecis.udel.edu

Dennis Pixton*

Mathematics, Binghamton University
Binghamton, NY 13902-6000
dennis@math.binghamton.edu

November 25, 1999

1 Introduction

In 1987 Tom Head [5] introduced the idea of *splicing* as a generative mechanism in formal language theory. This mechanism was popularized and extensively developed by Gheorghe Păun and his many coworkers, who recognized much of the subtlety and power of this mechanism and who used it as a foundation for the rapidly developing study of DNA based computing. See, for example, [11] and [12]. (The most necessary definitions are reviewed briefly in section 2.)

However, one of Head's most basic questions is still unanswered. It was established early that a finite splicing system would always produce a regular splicing language ([1], [13]), but not all regular languages can be generated in this way (a simple example is $(aa)^*$). Thus there remains the problem of precisely characterizing such splicing languages.

One of the first papers by Păun and his coworkers was [9], which introduced the notion of a *simple splicing system*. It is our goal here to give a simple description of the languages that result from a generalization of this notion, which we call a *semi-simple splicing system*. We are not interested in generalizing the results of [9], although our methods give simpler approaches even in the simple splicing case. Rather, we are interested in illustrating some approaches towards characterizing splicing languages in general.

Our main result is a characterization of semi-simple splicing languages in terms of certain directed graphs; this is carried out in sec-

*Research partially supported by DARPA/NSF CCR-9725021.

tion 3. Using this, we verify a conjecture for all splicing languages in this special case: All semi-simple splicing languages must have a *constant* word. Constant words have been used by Head in recent constructions of splicing systems, and in fact our result shows that most factors of a semi-simple splicing language are constant, from which we derive two further conclusions using Head's results: Semi-simple splicing languages are *strictly locally testable*, and they can be generated by *reflexive* splicing systems.

It was only recently shown that there are splicing languages that are not reflexive. It is still unknown whether there are splicing languages that have no constants. Some of these results are in the first author's Ph.D. thesis [3]; related results will appear in [4].

2 Review of Splicing Theory

We work over a finite alphabet A . A *splicing rule over A^** is a quadruple $r = (u, v; u', v')$ of strings in A^* . If x and y are strings in A^* then we say that the string z is a result of *splicing x and y using the rule r* if we can factor $x = x_1uvx_2$, $y = y_1u'v'y_2$ and $z = x_1uv'y_2$. If $I \subset A^*$ we write $r(I)$ for the set of all words z that can be obtained this way, with x and y in I .

A *splicing scheme* or *H scheme* is a pair $\sigma = (A, R)$ where A is the alphabet and R is a set of rules over A^* . We say σ is finite if R is a finite set.

Given a language I in A^* we define $\sigma(I)$ to be the union of all $r(I)$, where $r \in R$. We define *iterated splicing* as follows:

$$\sigma^0(I) = I, \quad \sigma^{k+1}(I) = \sigma(\sigma^k(I)) \cup \sigma^k(I), \quad \sigma^*(I) = \bigcup_{k=0}^{\infty} \sigma^k(I).$$

By a *splicing system* or *H system* we mean a pair (σ, I) where σ is a splicing scheme over A^* and $I \subset A^*$.

In this paper we shall require that both σ and I be **finite**. This is the original definition due to Head, but there have been many generalizations, which we shall not consider in this paper. We say L is a *splicing language* if $L = \sigma^*(I)$ for some finite splicing system (σ, I) .

Following [9] we define a *simple* splicing scheme to be one in which all rules have the form $(a, 1; a, 1)$ for some $a \in A$ (we use 1 to refer to the empty string). We generalize this by defining a *semi-simple* splicing scheme to be one in which all rules have the form $(a, 1; b, 1)$ for $a, b \in A$.

We say a language L is a *simple splicing language* if $L = \sigma^*(I)$ for a finite simple splicing system (σ, I) . Semi-simple and other varieties of splicing languages are defined similarly.

At the end of the paper we present an example of a semi-simple splicing language which is not a simple splicing language.

3 Arrow Graphs

We begin with a language $L \subset A^*$. We define $R(L)$ to be the set of all semi-simple rules $r = (a, 1; b, 1)$, for $a, b \in A$, such that $r(L) \subset L$. Let $\sigma = (A, R)$ be a splicing scheme for which $R \subset R(L)$. We shall consider the possibility that $L = \sigma^*(I)$ for some finite initial set I .

We augment A, L, σ with new symbols S and T not in A as follows: $\bar{A} = A \cup \{S, T\}$, $\bar{R} = R \cup \{(S, 1; S, 1), (T, 1; T, 1)\}$, $\bar{\sigma} = (\bar{A}, \bar{R})$, and $\bar{L} = SLT$. This makes no significant difference:

Claim 3.1 $\bar{\sigma}(\bar{L}) \subset \bar{L}$. Moreover, if $I \subset A^*$ then $\bar{\sigma}^*(SIT) = S\sigma^*(I)T$.

Proof. This is immediate, since the extra rules involving S and T essentially do nothing: If x, y are in XA^*Y then splicing x, y using $(S, 1; S, 1)$ produces y , and splicing x, y using $(T, 1; T, 1)$ produces x . \square

Now we develop a graphic representation for \bar{L} .

An *arrow* shall mean a triple $e = (a, w, a')$ in $\bar{A} \times A^* \times \bar{A}$. We normally write an arrow as $e = a \xrightarrow{w} a'$. Given such an arrow, we say a is the *initial vertex* of e and a' is the *terminal vertex* of e .

We define the *arrow graph* G of the pair (σ, L) to be the following directed graph: The vertex set of G is \bar{A} , and the arrow $a \xrightarrow{w} a'$ is an *edge* of G from a to a' if there is a rule $(a, 1; b, 1)$ in \bar{R} so that bwa' is a factor of \bar{L} . (A *factor* of a language M is a string y such that, for some strings x and z , $xyz \in M$.)

Note that G is an infinite graph, if L is infinite:

Claim 3.2 $S \xrightarrow{w} T$ is an edge of G if and only if $SwT \in \bar{L}$.

Proof. If $SwT \in \bar{L}$ then SwT is a factor of \bar{L} and $(S, 1; S, 1)$ is in \bar{R} , so $S \xrightarrow{w} T$ is an edge. Conversely, if $S \xrightarrow{w} T$ is an edge then there is a rule $(S, 1; b, 1)$ in \bar{R} and bwT is a factor of \bar{L} . The only rule in \bar{R} involving S is $(S, 1; S, 1)$, so $b = S$ and SwT is a factor of \bar{L} . Since $\bar{L} \subset SA^*T$ we conclude that $SwT \in \bar{L}$. \square

We now introduce some algebraic structure on the arrow graph. If e_1 and e_2 are arrows then we say e_1 and e_2 are *adjacent* if the terminal vertex of e_1 is the initial vertex of e_2 . If $e_1 = a_0 \xrightarrow{w_1} a_1$ and $e_2 = a_1 \xrightarrow{w_2} a_2$ are adjacent we define their *product*, written $e_1 \cdot e_2$, to be the arrow $a_0 \xrightarrow{w_1 a_1 w_2} a_2$.

Claim 3.3 (Closure) *If e_1, e_2 is an adjacent pair of edges of G then $e_1 \cdot e_2$ is an edge of G .*

Proof. We write $e_1 = a_0 \xrightarrow{w_1} a_1$ and $e_2 = a_1 \xrightarrow{w_2} a_2$. Then there are rules $r_0 = (a_0, 1; b_0, 1)$ and $r_1 = (a_1, 1; b_1, 1)$ in \bar{R} and words $x_0 b_0 w_1 a_1 y_1$ and $x_1 b_1 w_2 a_2 y_2$ in \bar{L} . Using r_1 we can splice these words to produce $x_0 b_0 w_1 a_1 w_2 a_2 y_2$ in \bar{L} . Now $b_0 w_1 a_1 w_2 a_2$ is a factor of \bar{L} and r_0 is a rule in \bar{R} , so $e_1 \cdot e_2 = a_0 \xrightarrow{w_1 a_1 w_2} a_2$ is an edge of G . \square

As usual, a *path* in G is a sequence $\pi = \langle e_1, e_2, \dots, e_n \rangle$ of edges so that each pair e_k, e_{k+1} is adjacent. In this case we write $e_k = a_{k-1} \xrightarrow{w_k} a_k$ and we say π is a path from a_0 to a_n . We use a nonstandard definition for the label of such a path:

$$\lambda(\pi) = a_0 w_1 a_1 w_2 \dots a_{n-1} w_n a_n.$$

Of course we identify a single edge e with the path $\langle e \rangle$, so $\lambda(e)$ is also defined.

On the other hand, the multiplication on edges is obviously associative, so we can define unambiguously the product $e_1 \cdot e_2 \cdot \dots \cdot e_n$ of the edges in a path π . The following is immediate from 3.3 and the definitions:

Claim 3.4 *If $\pi = \langle e_1, e_2, \dots, e_n \rangle$ is a path in G then $e = e_1 \cdot e_2 \cdot \dots \cdot e_n$ is an edge in G and $\lambda(e) = \lambda(\pi)$.* \square

Finally, we define the language of the graph G , written $L(G)$, to be the set of all labels of paths in G from S to T .

Claim 3.5 $L(G) = \bar{L}$.

Proof. According to 3.2, $\bar{L} \subset L(G)$. On the other hand, if π is a path in G from S to T then 3.4 produces an edge from S to T with the same label, so 3.2 also gives $L(G) \subset \bar{L}$. \square

Now our goal is to replace G with a finite subgraph which still represents \bar{L} . For this we require a way to factor long edges. First we have a very simple partial factorization:

Lemma 3.6 (Prefix edge) *If $a \xrightarrow{ua'v} a''$ is an edge of G and $a' \in A$ then $a \xrightarrow{u} a'$ is an edge of G .*

Proof. This is trivial: Since $a \xrightarrow{ua'v} a''$ is an edge there are a rule $r = (a, 1; b, 1)$ in \bar{R} and a word $z = xbuav'a''y$ in \bar{L} . The rule r and the factor bua' of z establish that $a \xrightarrow{u} a'$ is an edge of G . \square

We define an edge to be a *prime* edge if it cannot be written as the product of two edges. The following is proved by a standard induction on the length of the label of an edge:

Claim 3.7 *Every edge is the product of a sequence of prime edges.* \square

This factorization is unique. This is not necessary for our paper, so we omit the proof, but this uniqueness justifies our use of the term *prime edge* instead of *irreducible edge*.

We define the *prime arrow graph* G_0 to be the subgraph of G with the same vertex set but using only the prime edges of G . As an immediate consequence of 3.5 and 3.7 we have:

Claim 3.8 $L(G_0) = \bar{L}$.

The next result is the key observation about prime edges.

Claim 3.9 *Suppose $L = \sigma^*(I)$ for some subset $I \subset L$. If $a_0 \xrightarrow{w} a_1$ is a prime edge in G then w is a factor of I .*

Proof. For any edge $e = a_0 \xrightarrow{w} a_1$ there is a rule $(a_0, 1; b_0, 1)$ so that b_0wa_1 is a factor of $\bar{L} = \bar{\sigma}^*(\bar{I})$. Thus we can define an index $N(e)$ to be the minimal n such that, for some such rule, b_0wa_1 is a factor of $\bar{\sigma}^n(\bar{I})$. We shall prove the claim by induction on $N(e)$.

The claim is obviously true for all prime edges e with $N(e) = 0$. So let $e = a_0 \xrightarrow{w} a_1$ be a prime edge with $n = N(e) > 0$ so that the claim holds for all prime edges e' with $N(e') < n$. Then there are a rule $r_0 = (a_0, 1; b_0, 1)$ and a string z_0 in $\bar{\sigma}^n(\bar{I})$ which can be factored as $z_0 = x_0b_0wa_1y_1$. Since $n > 0$ there are a rule $r = (a, 1; b, 1)$ and strings

$z = xay'$ and $z' = x'by$ in $\bar{\sigma}^{n-1}(\bar{I})$ so that z_0 is the result of splicing z and z' using r . That is,

$$xay = x_0b_0wa_1y_1.$$

There are four cases, depending on where the “ a ” appears in z_0 :

Case 1: $|xa| \geq |x_0b_0wa_1|$: Then b_0wa_1 is a factor of xa , and hence of $z = xay'$, contradicting $N(e) = n$.

Case 2: $|x_0b_0| < |xa| < |x_0b_0wa_1|$: Then $w = uav$, so we have the prefix edge $e_1 = a_0 \xrightarrow{u} a$. Moreover, $y = va_1y_1$ so bva_1 is a factor of z' . Using this and the rule b we conclude that $e_2 = a \xrightarrow{v} a_1$ is an edge. Then the factorization $e = e_1 \cdot e_2$ contradicts primality.

Case 3: $|xa| = |x_0b_0|$: Then $y = wa_1y_1$. Thus $z' = x'by = x'bwa_1y_1$ so, using the rule r , we conclude that $e' = a \xrightarrow{w} a_1$ is an edge. Moreover, it is a prime edge. For if not then there are edges e_1 and e_2 with $e' = e_1 \cdot e_2$. We write $e_1 = a \xrightarrow{u} a'$ and $e_2 = a' \xrightarrow{v} a_1$, so $w = ua'v$. Then $e'_1 = a_0 \xrightarrow{u} a'$ is a prefix edge of e , so $e = e'_1 \cdot e_2$, contradicting primality of e .

Since $N(e') \leq n - 1$ the claim is true for e' , and so w is a factor of I .

Case 4: $|xa| < |x_0b_0|$: Then b_0wa_1 is a factor of y , and hence of $z' = x'by$, contradicting $N(e) = n$. \square

The following is our main result; it characterizes semi-simple splicing languages in terms of arrow graphs.

Theorem 3.10 *Suppose L is a language in A^* , R is a subset of $R(L)$, $\sigma = (A, R)$ and G_0 is the prime arrow graph for (σ, L) . Then there is a finite $I \subset A^*$ such that $L = \sigma^*(I)$ if and only if G_0 is finite.*

Proof. It is immediate from 3.9 that G_0 is finite if I is finite.

Conversely, suppose that G_0 is finite. For each prime edge $e = a_0 \xrightarrow{w} a_1$ and each rule $(a_0, 1; b_0, 1)$ in \bar{R} select, if possible, one word in \bar{L} which has b_0wa_1 as a factor. Let $\bar{I} = SIT$ be the set of these strings. We claim that $\bar{L} = \bar{\sigma}^*(\bar{I})$, and hence $L = \sigma^*(I)$.

To see this, take any edge e in G from S to T , and let $e = e_1 \cdot e_2 \cdot \dots \cdot e_n$ be its prime factorization, with $e_k = a_{k-1} \xrightarrow{w_k} a_k$. For $1 \leq k \leq n$ select a rule $r_k = (a_{k-1}, 1; b_{k-1}, 1)$ and a word z_k in \bar{I} with $b_{k-1}w_k a_k$ as a factor. Note that $a_0 = b_0 = S$, so w_1 is a prefix of z_1 . Similarly, w_n is a suffix of z_n . It is then clear that z_1, z_2, \dots, z_n can be spliced, using the rules r_2, \dots, r_n in this order, to produce $\lambda(e)$. \square

Corollary 3.11 *A language $L \subset A^*$ is a semi-simple splicing language if and only if the prime arrow graph constructed from $(\hat{\sigma}, L)$, with $\hat{\sigma} = (A, R(L))$, is finite.*

Proof. Just notice that if $L = \sigma^*(I)$ for any $\sigma = (A, R)$ with $R \subset R(L)$ then $L = \hat{\sigma}^*(I)$. \square

This corollary provides an algorithm for determining whether a regular language L is a semi-simple splicing language. First, to check whether $r = (a, 1; b, 1)$ is in $R(L)$ we have to decide whether $r(L) \subset L$, and this is algorithmically feasible since $r(L)$ is a regular language if L is regular; specifically, in terms of the quotient operation, $r(L) = (L(aA^*)^{-1})a((A^*b)^{-1}L)$. Further, if a, a' are in \bar{A} we define $E(a, a')$ to be the set of w such that $a \xrightarrow{w} a'$ is an edge of G . This is a regular language, since it can be written as the union, over b for which $(a, 1; b, 1) \in R(L)$, of $(A^*b)^{-1}L(aA^*)^{-1}$. Moreover, the set $E_0(a, a')$, defined as above but for edges of G_0 , is also regular, since it may be written as $E(a, a')$ minus the union of the products $E(a, a'')a''E(a'', a')$, and so it is algorithmically decidable whether $E_0(a, a')$ is finite.

For the applications in the next section we require the following:

Lemma 3.12 (Approximate factorization) *If G_0 is finite then there is an integer K with the following property: Suppose $e = a_0 \xrightarrow{xyz} a_1$ is an edge of G and $|y| \geq K$. Then there is a factorization $e = e' \cdot e''$ so that $e' = a_0 \xrightarrow{xu} a$ and $e'' = a \xrightarrow{vz} a_1$ (and hence $y = uav$).*

Proof. Let K_0 be the maximum length of the label of an edge of G_0 , and set $K = K_0 - 1$.

Take e as in the statement of the lemma and let $e_1 \cdot e_2 \cdot \dots \cdot e_n$ be its prime factorization. Let k be the last index for which $|\lambda(e_1 \cdot e_2 \cdot \dots \cdot e_k)| \leq |a_0x|$. By the definition of K , $|\lambda(e_1 \cdot e_2 \cdot \dots \cdot e_{k+1})| \leq |a_0xy|$. Then $e' = e_1 \cdot e_2 \cdot \dots \cdot e_{k+1}$ and $e'' = e_{k+2} \cdot \dots \cdot e_n$ have the desired properties. \square

4 Constants and Reflexivity

Schützenberger [14] defined a *constant* of a language L to be a string $c \in A^*$ so that, for all strings x, y, x', y' , we have xcy' in L if xcy and $x'cy'$ are in L . This has an obvious relation to splicing: c is a constant of L if and only if $r(L) \subset L$, where $r = (c, 1; c, 1)$. For simple splicing systems we can interpret the graph characterization of section 3 in terms of constants:

Theorem 4.1 *A language L in A^* is a simple splicing language if and only if there is K so that every factor of L of length at least K contains a symbol which is a constant of L .*

Proof. Let M be the set of all symbols in A which are constants of L , let \tilde{R} be the set of all rules $(a, 1; a, 1)$ with $a \in M$, and let $\tilde{\sigma} = (A, \tilde{R})$. Then clearly $L = \sigma^*(I)$ for some simple splicing scheme σ if and only if $L = \tilde{\sigma}^*(I)$. Hence, by Theorem 3.10, L is a simple splicing language if and only if the prime arrow graph G_0 constructed from $(L, \tilde{\sigma})$ is finite. However, an arrow $a \xrightarrow{w} a'$ is an edge if and only if awa' is a factor of \tilde{L} and $a = S$ or $a \in M$. Hence an edge $a \xrightarrow{w} a'$ is a prime edge if and only if w does not contain any elements of M . Thus G_0 is finite if and only if there is a bound, K_1 , on the lengths of factors of L that do not contain symbols in M , and the result follows with $K = K_1 + 1$. \square

It is unknown whether every splicing language must have a constant; there are regular languages which have no constants. In the semi-simple case there are many constants:

Theorem 4.2 *If L is a semi-simple splicing language then there is a constant K so that every string in A^* of length at least K is a constant of L .*

Proof. Let K be the number given by approximate factorization (3.12). Suppose that $c \in A^*$ has length at least K and suppose xcy and $x'cy'$ are in L . Then approximate factorization applied to $e = S \xrightarrow{x'cy'} T$ gives $e = e' \cdot e''$, where $e'' = a \xrightarrow{d'y'} T$ and $c = dad'$. Now $S \xrightarrow{xcy} T$ is an edge and $xcy = xdad'y$ so we have the prefix edge $\bar{e} = S \xrightarrow{xd} a$ (from 3.6). Hence $\bar{e} \cdot e'' = S \xrightarrow{xdad'y'} T$ is an edge, so $xdad'y' = xcy'$ is in L , as desired. \square

McNaughton and Papert [10] defined the notion of *strict local testability*, and de Luca and Restivo [2] translated this definition to the following: A language L is strictly locally testable if and only if there is K so that every string in A^* of length at least K is a constant. Thus, as in [8], we have:

Corollary 4.3 *If L is a semi-simple splicing language then L is strictly locally testable.* \square

A splicing scheme $\sigma = (A, R)$ is called *reflexive* if whenever a rule $(u, v; u', v')$ is in R then $(u, v; u, v)$ and $(u', v'; u', v')$ are also in R . This condition was introduced in [6] as a necessary condition for a splicing scheme to actually represent DNA recombination. It was recently discovered that there are splicing languages that cannot be generated by reflexive splicing systems. An example is the set of all words on $\{a, b\}^*$ that have at most two b 's. See [3] and [4].

Head has shown ([7, 8]) how the presence of sufficiently many constants in a language leads to a splicing structure for the language. Precisely, he proves that if L is a regular language and there is a finite set F of constants of L so that $L \setminus A^*FA^*$ is finite then L is a reflexive splicing language. Thus:

Corollary 4.4 *If (σ, I) is a semi-simple splicing system then there is a reflexive splicing system $(\hat{\sigma}, \hat{I})$ such that $\sigma^*(I) = \hat{\sigma}^*(\hat{I})$. \square*

We conclude with a very simple example of a semi-simple splicing language, illustrating our main results.

Let $A = \{a, b, c\}$, let R contain the single rule $(a, 1; b, 1)$, and let $I = \{abccab\}$. Following the arguments of 3.9 we can readily determine the prime arrow graph G_0 ; we find the following set of prime edges: $S \xrightarrow{abccab} T$, $S \xrightarrow{1} a$, $S \xrightarrow{abcc} a$, $a \xrightarrow{cc} a$, $a \xrightarrow{1} T$, $a \xrightarrow{ccab} T$. (There are other prime edges, but we shall ignore them, since they do not appear on any paths from S to T .) From this determination of G_0 it is easy to see that the splicing language is $L = \sigma^*(I) = \{a\} \cup \{a, ab\}(cca)^+\{1, b\}$. The reader may check that any rule $r = (x, 1; x, 1)$, with $x \in A$, may be applied to two copies of $abccab$ to produce a string which is not in L ; hence L is not a simple splicing language.

From the proof of Theorem 4.2 we see that every factor of L of length at least 8 is a constant of L , and we can follow the arguments of [7] to produce a reflexive splicing system whose language is L . In fact, our example is simpler: The shortest constant of L is cc , and this constant appears in all elements of L except the string a . Moreover, L is generated by a reflexive splicing scheme with only one rule, $(cc, 1; cc, 1)$.

References

- [1] K. Culik II and T. Harju. Splicing semigroups of dominoes and DNA. *Discrete Appl. Math.*, 31:261–277, 1991.

- [2] A. Deluca and A. Restivo. A characterization of strictly locally testable languages and its application to subsemigroups of a free semigroup. *Inform. and Control*, 44:300–319, 1980.
- [3] E. Goode. *Constants and Splicing Systems*. PhD thesis, Binghamton University, 1999.
- [4] E. Goode and D. Pixton. Syntactic monoids, simultaneous pumping, and H systems. In preparation.
- [5] T. Head. Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors. *Bull. Math. Biol.*, 49(6):737–759, 1987.
- [6] T. Head. Splicing systems and DNA. In G. Rozenberg and A. Salomaa, editors, *Lindenmayer Systems: Impacts on Theoretical Computer Science, Computer Graphics and Developmental Biology*, pages 371–383. Springer Verlag, Berlin, Heidelberg, New York, 1992.
- [7] T. Head. Splicing languages generated with one sided context. In G. Paun, editor, *Computing With Bio-molecules—Theory and Experiments*, pages 269–282. Springer-Verlag, Singapore, 1998.
- [8] T. Head. Splicing representations of strictly locally testable languages. *Discrete Applied Mathematics*, 87:139–147, 1998.
- [9] A. Mateescu, G. Păun, G. Rozenberg, and A. Salomaa. Simple splicing systems. *Discrete Applied Mathematics*, 1996.
- [10] R. McNaughton and S. Papert. *Counter-Free Automata*. MIT Press, 1971.
- [11] G. Păun. On the power of the splicing operation. *International Journal of Computer Mathematics*, 59:27–35, 1995.
- [12] G. Păun, G. Rozenberg, and A. Salomaa. Computing by splicing. *Theoretical Computer Science*, 168(2):321–336, 1996.
- [13] D. Pixton. Regularity of splicing languages. *Discrete Appl. Math.*, 69(1–2):99–122, August 1996.
- [14] M.P. Schützenberger. Sur certaines opérations de fermeture dans les langages rationnels. *Symposia Math.*, 15:245–253, 1975.