Splicing Systems: Regularity and *Below*

Tom Head[1], Dennis Pixton[1], and Elizabeth Goode[2]

[1] Department of Mathematical Sciences,
Binghamton University
Binghamton, New York USA
[2] Mathematics Department
Towson State University
Towson, Maryland, USA

Abstract

The motivation for the development of splicing theory is recalled. Attention is restricted to *finite* splicing systems, which are those having only finitely many rules and finitely many initial strings. Languages generated by such systems are necessarily regular, but not all regular languages can be so generated. The splicing systems that arose originally, as models of enzymatic actions, have two special properties called *reflexivity* and *symmetry.* We announce the Pixton-Goode procedure for deciding whether a given regular language can be generated by a finite *reflexive* splicing system. Although the correctness of the algorithm is not demonstrated here, two propositions that serve as major tools in the demonstration are stated. One of these is a powerful pumping lemma. The concept of the syntactic monoid of a language provides sharp conceptual clarity in this area. We believe that there may be yet unrealized results to be found that interweave splicing theory with subclasses of the class of regular languages and we invite others to join in these investigations.

1 The original motivation for the splicing concept

The splicing concept was developed in the 1980's [12] following the first author's study of the first edition of B. Lewin's beautiful book *Genes* [17]. The *sequential* feature of the biological macromolecules was used to treat these molecules as material realizations of abstract character strings. The nucleic acids, proteins, and many additional polymers admit such string models. However, the detailed nature of the splicing concept arose from considerations of the cut & paste activity made possible through the action of restriction enzymes on double stranded DNA molecules (dsDNA). There are currently more than 200 different restriction enzymes commercially available. These enzymes cut dsDNA at one covalent bond of each of the two sugar-phosphate backbones occurring in sub-segments having specific sequences. Such cuts sever the molecule leaving two freshly cut ends that have the potential, in the presence of a ligase enzyme, to be joined with appropriately matching ends of the same or other DNA molecules. For a reader who is not familiar with the derivation of the abstract model of splicing from the biochemical processes that the model idealizes, we recommend reading the explanation given in [12], [15], or [20]. The original formalism for splicing systems [12] was rigidly derived from the biochemical processes being simulated. For thinking about models of molecular processes, there is still value in the original formalism. However, for proving formal theorems at a less restricted level of generality, the formal definition of splicing used here, which is essentially Gh. Paun's, has become standard.

Let A be a finite set to be used as an alphabet. Let A^* be the set of all strings over A. By a language we mean a subset of A^*. A *splicing rule* is an element $r = (u, v', u', v)$ of the product set $(A^*)^4$. A splicing rule r *acts on a language* L producing the language $r(L) = \{xuvy$ in $A^* : L$ contains strings $xuv'q$ & $pu'vy$ for some q, p in $A^*\}$. For each set, R, of splicing rules we extend the definition of $r(L)$ by defining $R(L) = \cup\{r(L) : r$ in $R\}$. A rule r *respects* the language L if $r(L)$ is contained in L and a set R of rules *respects* L if $R(L)$ is contained in L. By the *radius* of a splicing rule (u, v', u', v) we mean the maximum of the lengths of the strings u, v', u', v.

Definitions. A *splicing scheme* is a pair $\sigma = (A, R)$, where A is a *finite* alphabet and R is a *finite* set of splicing rules. For each language L and each non-negative integer n, we define $\sigma^n(L)$ inductively: $\sigma^0(L) = L$ and, for each non-negative integer k, $\sigma^{k+1}(L) = \sigma^k(L) \cup R(\sigma^k(L))$. We then define $\sigma^*(L) = \cup\{\sigma^n(L) : n \geq 0\}$. A *splicing system* is a pair (σ, I), where σ is a splicing scheme and I is a *finite* initial language contained in A^*. The *language generated by* (σ, I) is $L(\sigma, I) = \sigma^*(I)$. A language L is a *splicing language* if $L = L(\sigma, I)$ for some splicing system (σ, I).

A rule set R is *reflexive* if, for each rule (u, v', u', v) in R, the rules (u, v', u, v') and (u', v, u', v) are also in R. A rule set R is *symmetric* if, for each rule (u, v', u', v) in R, the rule (u', v, u, v') is in R. When R is reflexive or symmetric we say the same of any scheme or system having R as its rule set. Reflexivity and symmetry are inherent features of splicing systems as defined originally in [12]. In fact, splicing systems that model the cut & paste action of restriction enzymes and a ligase are necessarily reflexive and symmetric as is easily confirmed by envisioning the activity of the enzymes and DNA molecules in solution. Consequently, from a modeling perspective, the most important splicing systems are those that are reflexive and symmetric.

The motive for the introduction of the formal splicing concept was the establishment of a passageway between formal language theory and the biomolecular sciences. The most secure prediction was that formal representations of enzymatic actions would provide a novel stimulation for the development of language theory. This prediction has been confirmed in the later chapters of [20] and by continuing developments in progress by many theoretical computer scientists. The less secure prediction was that the development of formal theory would eventually yield results of value to biomolecular scientists. One might hope, for example, that the demonstration of the regularity of splicing languages will eventually be represented in software that accepts a list of enzymes and the sequence data for a list of DNA molecules and decides whether a specified DNA molecule could arise through the action of the specified enzymes on the DNA molecules in the given list. The long-range hope has been that splicing theory would be the initial step in the development of a much broader approach to the modeling of important enzymatic processes using language theory.

2 The regularity of splicing languages

Can finite splicing systems generate only a very restricted class of languages? This was the first question asked. Splicing theory, as defined both here and originally, is concerned with *sets* of strings over an alphabet, not *multi-sets*. One of the earliest results on splicing systems [9] showed that *if* splicing theory were interpreted to deal with multi-sets then the action of each Turing machine could be simulated by the action of an appropriate splicing system. This result undermined confidence that splicing languages are always regular. Fortunately it was quickly announced [6] [7] that splicing languages *are* always regular. A later proof [22] gave an explicit construction followed by an induction on an insightfully specified inductive set. A slight reformulation of this proof appears in Chapter 6 of [20]. It had been noted early that not all regular languages are splicing languages [10]. That the regular languages $(aa)^*$ and $a^*ba^*ba^*$ are not splicing languages is easily confirmed. So, which regular languages are splicing languages? We would like to have a beautiful theorem that identifies the splicing languages with some crucial previously known class of regular languages, or at least some closely related class. As yet we have no such characterization. It is easily confirmed that every strictly locally testable (SLT) language is a splicing language [12]. (See [19] and [8] for the definition of SLT languages.) It is also known from examples [12] that even splicing languages that arise as explicit models of DNA behavior may fail to be SLT. The language $b(aa)^*$ is an abstract example of a splicing language that is neither SLT nor even aperiodic. (See [21] for the definition of aperiodic.)

With no crisp characterization of the class of splicing languages as yet found, concern turned to the search for an algorithm for deciding whether a given regular language can be generated by a splicing system. There is, of course, an easily described procedure that is guaranteed to *discover* that a regular language L is a splicing language *if* L is a splicing language: For each positive integer n, for each set R of rules of radius $\leq n$, and for each subset I of L consisting of strings of length $\leq n$, decide whether $L(\sigma, I) = L$, where $\sigma = (A, R)$. Since both L and each such $L(\sigma, I)$ are regular, all these steps can be carried out. The procedure terminates when a system $L(\sigma, I)$ is found, but *fails to terminate* when L is *not* a splicing language. From this triviality, however, it follows that an algorithm will become available immediately if, for each regular language L, a bound, $N(L)$, can be calculated for which it can be asserted that L cannot be a splicing language unless there is a splicing system having rules of radius $\leq N(L)$ and initial strings of length $\leq N(L)$. We announce here such an algorithm, but we give only a skeleton of hints as to the justification of the algorithm. A complete treatment will soon be available by the latter two authors of the present article.

3 The Pixton-Goode Algorithm

Let L be a regular language and let $M = (Q, I, F)$ be the minimal deterministic automaton recognizing L, where Q, I & F are the sets of *all* states, the *initial* states, & the *final* states of M, respectively. We denote the state entered when M is in state q and the string x is read by qx. A procedure for deciding whether L is a reflexive splicing language will be outlined after providing justifying comments for three observations:

(a) We can decide whether a given splicing rule r *respects* the regular language L.
(b) We can adequately specify the set of *all* splicing rules that preserve L.
(c) We can compute an upper bound for the *radii* of the required splicing rules.

The reflexivity condition is required only to obtain (c). E. Goode proved in [11] that the regular language $a^*ba^*ba^* \cup a^*ba^* \cup a^*$ is a splicing language and also that it cannot be generated by any reflexive splicing system. Thus the reflexivity condition used in justifying (c) is significant. However, since the underlying molecular cut & paste activities modeled by splicing inevitably yield reflexive (and also symmetric) systems, this restriction does not seem severe.

(a) Observe that the rule $r = (u, u', v', v)$ respects $L = L(M) = L(Q, I, F)$ if and only if, for each ordered pair of states p, q of M, whenever $L(Q, \{puu'\}, F)$ and $L(Q, \{qv'v\}, F)$ are not empty, $L(Q, \{pu\}, F)$ contains $\{vx : x \text{ in } L(Q, \{qv'v\}, F)\}$.

(b) The syntactic congruence relation, C, in A^* is defined by setting uCv if and only if, for every pair of strings x & y in A^*, either xuy and xvy are *both* in L or *neither* is in L. Since L is regular, the number of C-congruence classes is a positive integer denoted here as $n(L)$. Suppose that a rule (u, u', v', v) respects L and that $uCu"$. It follows that $(u", u', v', v)$ also respects L: Whenever a pair $xu"u'y$ & $wv'vz$ is in L, then by the definition of C, so is the pair $xuu'y$ & $wv'vz$. Then $xuvz$ is in L and, by the definition of C, so is $xu"vz$, which confirms that $(u", u', v', v)$ respects L. This argument works in each of the four locations. Consequently every rule in the set of rules $\{(w, x, y, z) : wCu, xCu', yCv', zCv\}$ respects L if and only if any single rule in the set respects L. (This observation, which establishes a provocative link between syntactic monoids and splicing, has been recorded independently in [11] and in [3] where it appears as Proposition 9.3.) From each of the $n(L)^4$ quadruples of syntactic classes determined by L in A^*, we choose one rule and test it as in (a) to determine whether it respects L. Each congruence class is itself a regular language, consequently, for each non-negative integer k, we can list all the strings of length at most k in the class. This allows us, for each non-negative integer k, to list, in a conceptually coherent manner, *all* rules of radius $\leq k$ that preserve L.

(c) It is sufficient to consider splicing rules of radius not greater than $N = 2(n(L)^2 + 1)$.

Since assertion (c) requires much detailed work, its full justification must await the forthcoming article by the latter two authors. Here we state only the two major intermediate results from which the justification is constructed. The first of these is a Lemma that plays a crucial role in intricate string calculations:

Two-Sided Pumping Lemma. *Let L be a regular language over an alphabet A. For each string w in A^*, having length greater than $n(L)^2$, there is a factorization $w = xyz$ with y non-null for which, for every non-negative integer k, and every quadruple of strings p, q, s, t in A^*,*

$$\begin{array}{lll} pxq \text{ is in } L & \text{if and only if} & pxy^kq \text{ is in } L\text{; and} \\ sxt \text{ is in } L & \text{if and only if} & sy^kzt \text{ is in } L. \end{array}$$

The second required tool is a Proposition that was proved earlier [11] where it played a crucial role in answering previous splicing questions:

Proposition. *A regular language L is a reflexive splicing language if and only if there is a finite reflexive set, R, of splicing rules for which $R(L)$ is contained in L and $L\backslash R(L)$ is finite.*

If L is a splicing language $L(\sigma, I)$ with $\sigma = (A, R)$ then the only the strings in L that can fail to lie in $R(L)$ are those in I and consequently $L\backslash R(L)$ is finite. Thus necessity is trivial. When $L\backslash L(R)$ is finite, one might hope $L = L(\sigma, L\backslash L(R))$ with $\sigma = (A, R)$. Although this is not in general the case, by using the assumed reflexivity of R, the finite sets R and $L\backslash L(R)$ can be finitely enlarged to produce sets R' and I', respectively, for which $L = L(\sigma', I')$ with $\sigma' = (A, R')$ as demonstrated in [11].

With each regular language L and each positive integer k we associate the following reflexive set of splicing rules:

$T_k = \{(u, u', v', v)$: the radius of (u, u', v', v) is $\leq k$ and L is preserved by each of the three rules (u, u', v', v), (u, u', u, u') and $(v', v, v', v)\}$.

Theorem. *A regular language L is a reflexive splicing language if and only if $L\backslash T_k(L)$ is finite where $k = 2(n(L)^2 + 1)$.*

Recall that $n(L)$ is the number of syntactic congruence classes of A^* determined by L and that, from (a) and (b) above, $T_k(L)$ is algorithmically constructible. Consequently the Theorem assures that, since the finiteness of $L\backslash T_k(L)$ can be decided, it can be decided whether L is a reflexive splicing system.

4 Room at the bottom

An extensive literature exists relating various extensions of the splicing system concept with universal computational schemes as exposited in [20]. Such extensions were motivated by the desire to find additional new models for biomolecular computation following L. Adleman's wet-lab computation [1]. Many splicing theorists have regarded *finite* splicing systems as an impoverished level of the theory. However, *when the motive is to model enzymatic processes, then it is a joy when one finds that extremely simple systems are adequate models.* The study of sub-classes of the regular languages is an intensely algebraic theory that is well developed [19] [21] [23] [2]. The syntactic congruence is a fundamental tool in this literature and it has greatly clarified the work we have reported here. Can more extensive interactions be found between this literature and the study of *restricted* types of finite splicing systems? We hope so and we recommend that the interested reader join in this search. In [18] the class of *simple splicing systems* was introduced and studied. This work motivated a detailed re-investigation in [13] of the null-context splicing systems, which were introduced originally in [12]. The null-context level has also been examined recently in relation to the naturally occurring DNA restructuring carried out by ciliates [16]. Progressively less simple splicing systems have been defined and studied in [14] and in [11], which also includes the solution of the open problem proposed in [14]. We believe there is more room for worthwhile research at the *bottom* of splicing theory.

Late Breaking News. On arrival at DNA-8, the first author was delighted to be given by G. Mauri a copy of [3], which contains one of the most provocative observations made in the present article: the connection between syntactic monoids and splicing. More recently, C. De Felice has forwarded [5] to us. This mutual interest in *finite* splicing is especially encouraging. The reader who finds the present article of interest will surely wish to see these 'BFMZ' references and the additional references they contain, such as [4].

Acknowledgments. The first author is exceedingly grateful for the invitation from Masami Hagiya to speak at the *8th Workshop on DNA Computers.* All three authors profited at various intervals during the previous decade from the support of their research through the NSF grants CCR-9201345, CCR-9509831 and by a subcontract through Duke University of the DARPA/NSF CCR-9725021 research program headed by John Reif. This support is gratefully acknowledged.

References

1. L. Adleman, Molecular computation of solutions of combinatorial problems, *Science* **266**(1994)1021-1024.
2. J. Almeida, *Finite Semigroups and Universal Algebra*, World Scientific, Singapore (1994).
3. P. Bonizzoni, C. De Felice, G. Mauri, R. Zizza, On the power of linear and circular splicing systems, (submitted).
4. P. Bonizzoni, C. De Felice, G. Mauri, R. Zizza, Decision problems for linear and circular splicing, DLT2002 volume of *LNCS* (to appear).
5. P. Bonizzoni, C. De Felice, G. Mauri, R. Zizza, The structure of reflexive splicing languages via Schutzenberger constants, (submitted).

6. K. Culik II, T. Harju, The regularity of splicing systems and DNA, in: *Proc. ICALP '89, LCNS* **372**(1989)222-233.
7. K. Culik II, T. Harju, Splicing semigroups of dominoes and DNA, *Discrete Appl. Math.* **31**(1991)261-277.
8. A. DeLuca, A. Restivo, A characterization of strictly locally testable languages and its application to subsemigroups of a free semigroup, *Inform. and Control* **44**(1980)300-319.
9. K.L. Denninghoff, R. Gatterdam, On the undecidability of splicing systems, *Inter. J. Computer Math.* **27**(1989)133-145.
10. R.W. Gatterdam, Splicing systems and regularity, *Inter. J. Computer Math.* **31**(1989)63-67.
11. T.E. Goode Laun, *Constants and Splicing Systems*, Dissertation Binghamton University, Binghamton, New York (1999).
12. T. Head, Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors, *Bull. Math. Biology* **49**(1987)737-759.
13. T. Head, Splicing representations of strictly locally testable languages, *Discrete Appl. Math.* **87**(1990)139-147.
14. T. Head, Splicing languages with one-sided context, in: Gh. Paun, Ed., *Biomolecular Computing - Theory and Experiment*, Springer-Verlag (1998)269-282.
15. T. Head, Gh, Paun, D. Pixton, Language theory and molecular genetics: generative mechanisms suggested by DNA recombinations, Chapter 7, Vol. 2 of: G. Rozenberg & A. Salomaa, Eds., *Handbook of Formal Languages*, Springer, Berlin (1997)295-360.
16. J. Kari, L. Kari, Context-free recombination, in: C. Martin-Vide & V. Mitrana, Eds., *Where Mathematics, Computer Science, Linguistics and Biology Meet*, Kluwer Academic Pub., Dordrecht (2001).
17. B. Lewin, *Genes*, (1st. ed.) Wiley, New York (1983).
18. A. Mateescu, Gh. Paun, G. Rozenberg, A. Salomaa, Simple splicing systems, *Discrete Appl. Math.* **84**(1996)145-163.
19. R. McNaughton, S. Papert, *Counter-free Automata*, MIT Press, Cambridge, MA (1971).
20. Gh. Paun, G. Rozenberg, A. Salomaa, *DNA Computing - New Computing Pardigms*, Springer-Verlag, Berlin (1998).
21. J.E. Pin, *Varieties of Formal Languages*, Plenum Pub. Co. (1986).
22. D. Pixton, Regularity of splicing systems, *Discrete Appl. Math.* **69**(1996)101-124.
23. H. Straubing, *Finite Automata, Formal Logic, and Circuit Complexity*, Birkhauser, Boston, MA (1994).