Separating some splicing models *

P. Bonizzoni, C. Ferretti, G. Mauri, R. Zizza

Dipartimento di Informatica Sistemistica e Comunicazione

Università degli Studi di Milano-Bicocca

via Bicocca degli Arcimboldi 8, 20136 Milano - ITALY

contact: ferretti@disco.unimib.it

Abstract

In this paper we show that the three main definitions of the splicing operation known in the literature, i.e. Head [1], Paun [2] and Pixton [4] definitions, give rise to different subclasses of regular languages, when a finite set of rules is iterated on a finite set of axioms. More precisely, we show that the family of regular languages generated by finite splicing as defined in the early paper by Head, is strictly included in the family defined later by Paun, which is in turn strictly included in the splicing family defined by Pixton. We describe instance languages in the difference sets, and we prove how they cannot be generated by the smaller families.

Keywords: Formal Languages, Molecular Computing, Splicing systems

*A preliminary version of this paper appeared in the Proceedings of Grammar Systems 2000, Austria

1 Introduction

In 1987 Tom Head [1] defined splicing systems to model the recombinant action of restriction enzymes and ligands on DNA molecules. Whenever a restriction enzyme finds a specific pattern along the double helix of a DNA molecule, it is able to *cut* the molecule inside the pattern, thus leaving two molecules with staggered or sticky ends. On the other hand, ligands are able to tie together molecules previously cut by restriction enzymes, producing a new molecule. This paste phenomenon is possible only under some chemical conditions: two pieces of DNA are pasted together if they have complementary sticky ends (in the Watson-Crick model, the four bases A,C,G,T constituting DNA are such that A is complementary to T and C is complementary to G) or no sticky ends.

Head translated the biological phenomenon described above, called splicing operation, in the framework of formal languages. In his formalism triples (α, μ, β) of strings define the pattern $\alpha\mu\beta$ searched by the restriction enzyme. In particular, μ is called the *crossing* and it defines the factor of the pattern that two molecules must have in common to be spliced and tied together. Paun (see for instance [2]) defined a slightly different formalism for patterns and the splicing operation, and one more definition of splicing was given by Pixton [4]. In the next section we will give formal definitions and some comments enlightening different behaviours of the three approaches.

In order to analyze the cut and paste phenomenon of the splicing operation in a formal language framework, an initial set of strings (initial set of DNA molecules) and an initial set of splicing rules (set of enzymes) are given. Thus the set of all possible molecules generated in the test tube is nothing but the language (called splicing language) of all possible strings that may be obtained by applying the rules to the strings. A splicing system is defined by giving the initial set of strings (or axioms) and the set of rules (or triples, according to which definition we use). Several splicing systems have been proposed in the literature [3], but here we do not take into account the bidimensional nature of DNA, we give no hypothesis on the number of copies in the initial set of molecules and we consider only a finite set of initial strings and a finite set of rules (or triples). Pixton showed that a class of languages generated under these hypotheses is a subclass of the class of regular languages [4]: such inclusion is proper simply by showing that the language $(a^2)^*$ cannot be generated by splicing. Moreover, the same result (regularity of finite splicing operation) holds for the other two definitions, since it is quite simple to check that given their respective definitions, Head splicing languages are always Paun splicing languages, and these are always Pixton splicing languages. Until now, it was unknown whether the inclusions among the above classes of languages were strict or not. In this paper we give a positive answer to this question, providing regular languages that separate the three classes.

2 Definitions

We recall the main formal definitions of splicing operation from the papers [1], [2] and [4]. In the following, A is the alphabet (finite set of symbols), A^* is the free monoid on A and ϵ is the empty word.

A **Head splicing system** [1] is defined as the tuple $S_H = (A, I, B, C)$, for $I \subset A^*$ a finite set of initial strings, B and C finite sets of triples, called *patterns* (α, μ, β), with $\alpha, \beta, \mu \in A^*$. The string μ of a triple is also called *crossing* of the triple. Given two words $u\alpha\mu\beta v, p\alpha'\mu\beta' q \in A^*$ and two patterns (α, μ, β) and (α', μ, β') that have the same crossing and are both in B or both in C, the splicing operation produces $u\alpha\mu\beta' q$ and $p\alpha'\mu\beta v$.

Observe that crossings define an equivalence relation among the triples. Thus, the string μ in the pattern directly creates the relation allowing to paste the strings. For instance, one could not design two distinct groups of patterns, whose generated strings must not be cross-pasted, if all these patterns need to have the identical empty crossing μ. Clearly, this fact restricts the power of the splicing operation, as we will see by giving a language that cannot be generated under this definition of splicing since it needs distinct groups of patterns with empty crossing.

A **Paun splicing system** [2] is a tuple $S_{PA} = (A, I, R)$, with I a finite set of initial strings over A and R a finite set of rules of the form $u'|u''\$v'|v''$, being $u', u'', v', v'' \in A^*$ and $|, \$$ symbols not in A. Given two strings $w'u'u''w'', z'v'v''z'' \in A^*$, the rule $u'|u''\$v'|v''$ produces (by splicing) the strings $w'u'v''z'', z'v'u''w''$.

Note that by Paun formalism for splicing systems patterns are pairs which define a binary relation; strings are allowed to tie together if they contain patterns which are in this relation. The relation among patterns could be, generally, any binary relation. This fact effectively links in a very strong way the two steps of cutting and pasting, since now the splicing system could be defined in such a way that the relation between two patterns means that when they are cut, the generated strings are immediately pasted together with no interference from strings cut from some other pattern: we just relate each pattern with only one pattern, independently from their content. This approach is very powerful, and it has often been chosen in the literature on splicing systems.

A **Pixton splicing system** [4] is a tuple $S_{PI} = (A, I, R)$, where $I \subset A^*$ is a finite initial language (the initial set of axioms), and R is a finite set of rules, where each rule r is a triple (α, β, α'). Given two strings $x_1 = u\alpha w$ and $x_2 = v\ \alpha' z$, then the application of rule r to x_1 and x_2 produces the strings $x_3 = u\beta z$ and $x_4 = v\beta w$.

Note that the cutting step and the pasting step are again operated one immediately after the other, but it is even allowed to *substitute* part of the pattern before pasting the strings originated by the cut.

Let us now give the formal definition of splicing language.

For a given splicing system S_X, with $X \in \{H, PA, PI\}$ and set of rules R, we denote $(x, y) \vdash_r w', w''$ the fact that the two strings w', w'' are produced from (or spliced by) x, y by using a rule r (or triples). Given a language $L \subseteq A^*$, we denote by $\sigma(L) = \{w', w'' \in A^* \mid (x,y)\vdash_r (w', w''),$ for $x, y \in L$ and $r \in R\}$. Note that x and y are not necessarily distinct. Thus, we define $\sigma^0(L) = L$, $\sigma^{i+1}(L) = \sigma^i(L) \cup \sigma(\sigma^i(L))$, $i \geq 0$, and then $\sigma^*(L) = \bigcup_{i\geq 0} \sigma^i(L)$. Given a splicing system S_X, with $X \in \{H, PA, PI\}$, the language $L(S_X) = \sigma^*(I)$ is the language generated by the splicing system S_X, I being the initial language in S_X. Roughly speaking, a splicing language is the smallest language obtained by iterated application (w.r.t. a definition) of the rules, starting with I. A language L is X generated (or L is a X *splicing language*) if a splicing system S_X exists such that $L = L(S_X)$.

3 Inclusion results

In this section we firstly present a non empty class of regular languages that are Paun splicing languages, but that can be generated by Head splicing systems only if the crossing of some triples is the empty word. By this fact, it follows that this class is not closed under union by using Head definition, while we can show that it is closed under union w.r.t. Paun definition.

Example 3.1 *Let* $A_1 = \{a, b, c, h\}$. *We define* $L_{abch} = \{a^2c^*a, ca^2c^*ac, bac^*a, ah, bh\}$, *depicted in Figure* 1. *Similarly, we can define* L_{defg}, L_{mnop} *regular languages on* $A_2 = \{d, e, f, g\}$ *and* $A_3 = \{m, n, o, p\}$, *respectively, where* $L_{abch}, L_{defg}, L_{mnop}$ *are three isomorphic languages. Let us consider the following Paun splicing system* $S_{PA} = (A_1, I, R)$, *for* $I = \{a^2c^na,\ bh, bac^na,\ ca^2c^nac, ah\}$, *where* $0 \leq n \leq 2$, *and* $R = \{ba|c\$bac|c, ca^2|c\$ca^2c|c, b|a\$a|h\}$. *We leave to the reader the proof that* S_{PA} *generates* L_{abch}. *Similarly, we can define the Paun splicing system generating* L_{defg} *and* L_{mnop}.

Let $L = L_{abch} \cup L_{defg} \cup L_{mnop}$ be a regular language over the alphabet $A_1 \cup A_2 \cup A_3$, as defined above. We have the following result.

Lemma 3.1 *The language* $L = L_{abch} \cup L_{defg} \cup L_{mnop}$ *is generated by a Paun splicing system and cannot be generated by a Head splicing system.*

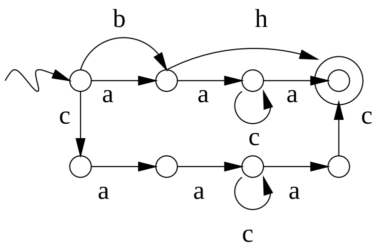

Figure 1: *The Paun splicing language* L_{abch}.

PROOF : We first give the Paun splicing system which generates L. Since $L_{abch}, L_{defg}, L_{mnop}$ are over distinct alphabets, it is immediate to obtain the Paun splicing system for generating L by union of the three Paun splicing systems given in example above.

Let us show that L is not generated by a Head splicing system. The proof is by contradiction. Thus let us assume to the contrary that $L = L(S_H)$ for some splicing system $S_H = (A, I, B, C)$. Now, given an arbitrary string $s_j = a^2c^ja \in L$, with j arbitrarily large, there are a pair p_1, p_2 of patterns and two strings w, u from which s_j is derived. We will assume that j is larger than any pattern in $B \cup C$. Assume that $p_1 = (\alpha, \mu, \beta)$ and $p_2 = (\delta, \mu, \gamma)$. We show in the following that μ cannot be a not empty crossing. In fact, assume that $\mu \neq \epsilon$. It is not restrictive to assume that $s_j = u_1\alpha\mu\gamma v_2$. Since $|\mu| < j$, otherwise j is not larger than any pattern in $B \cup C$, it follows that $\mu = a^ic^k$ or $\mu = c^ka^t$ for $0 \leq i \leq 2$, $0 \leq k < j$ and $0 \leq t \leq 1$, being $s_j \in L$. Thus the following two cases must be considered.

Case i: Let μ contain a c letter. It follows that p_1 is also a substring of a string x, where $x = ca^2c^mac$; thus, by splicing with patterns p_1 and p_2, a prefix of x can be combined with a suffix of a^2c^*a. But, then p_1 and p_2 generate a string not in L, a contradiction.

Case ii: Assume that μ contains only a letters. Again, if p_1 is a substring of string ca^2c^nac, we obtain as in case i a contradiction. Thus, let us assume that p_1 is not a substring of any string in $\{ca^2c^*ac\}$. Then it must be that $p_1 = (\epsilon, a, h)$. If $p_1 = (\epsilon, a, h)$, then $p_2 = (\delta, a, ac^sa^l)$, for $0 \leq l \leq 1$,

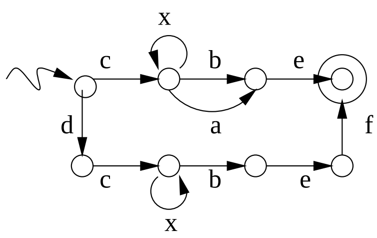

Figure 2: *A Pixton splicing language which is not a Paun splicing language.*

$0 \leq s \leq j$. But, it is immediate to verify that p_1 and p_2 generate also strings in $\{a^2c^*ac\}$ which are not in L, a contradiction.

Case i and ii together show that μ must be empty. Similarly, as above, we can show that given an arbitrary string $z_j = d^2f^jd \in L$, with j arbitrarily large, then z_j is derived by using a pair of patterns p_3, p_4 whose crossing must be empty. Clearly, by definition of language L, we need to keep the patterns with empty μ of the three alphabets separated, but we can only separate two groups: one in B and one in C. Two patterns with the same μ, but one from B and the other from C, will not interfere pasting together the strings they cut. We can suppose to put patterns over alphabets A_1 and A_2 in B, and patterns over A_3 in C. Consequently, for the strings in L considered above pattern p_1 can be used with pattern p_3 or p_4 to derive strings which contain letters from both alphabets A_1 and A_2, and hence which are not in L.

Now, we consider a regular language which is generated by a Pixton splicing system but which cannot be generated by a Paun splicing system, as proved in the lemma below.

Example 3.2 *Consider the regular language $L' = \{cx^*ae, cx^*be, dcx^*bef\}$, depicted in Figure 2. Let us consider the following Pixton splicing system $S_{PI} = (A, I, R)$, where $I = \{cae, cxae, dcbef, dcxbef\}$,* *and* $R = \{r_1 = (a, xa, xa), r_2 = (dcx, dcx, dc), r_3 = (a, b, a)\}$. *We can sketch that* $L' = L(S_{PI})$. *Indeed, by the first rule* r_1 *applied to two copies of the initial string* $cxae$, *and by repeated application of it, it is possible to generate the language* $L'' = \{cxx^*ae\}$. *Similarly, rule* r_2 *is used to generate the strings in* $\{dcx^*xbef\}$, *starting from the initial string* $dcxbef$. *Now, the last group of strings,* $\{cx^*be\}$ *is generated by the application of the rule* r_3 *to strings in* L'', *substituting letters* a *with letters* b. *The letter* a *and the string* dc *forbid the use of the rules to mix strings from the three different families in* L': *no other string is generated.*

We now prove that the language L' considered in the previous example cannot be a Paun splicing language.

Lemma 3.2 *The regular Pixton splicing language* $L' = \{cx^*ae, cx^*be, dcx^*bef\}$ *cannot be generated by a Paun splicing system.*

PROOF : Let us make the following observation. Any Paun splicing system, with finite sets of axioms and rules, could only develop the x^* loops putting in the rules some substrings leading to or trailing from x^*. But there's no way to choose those substring and avoid the cross-over between strings in $\{cx^*be\}$ and strings in $\{dcx^*bef\}$. The trick of first developing x^*a loops, and then *substituting* the a's with b's, which avoids such cross-overs, is feasible only in Pixton splicing systems.

Lemmas 3.1 and 3.2 prove the strict inclusion between Head splicing languages, Paun splicing languages and Pixton splicing languages: these are the main classes of languages studied in the framework of splicing systems. It could be quite interesting and useful to know whether an analogous result between splicing languages generated by using an initial language and a set of rules that are not finite still holds.

4 Acknowledgments

This work has been partially supported by MURST Project *"Unconventional Computational Models: Syntactic and Combinatorial Methods"* - *"Modelli di calcolo innovativi: Metodi sintattici e combinatori"*.

References

[1] T. Head, *Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors*, Bull. Math. Biology, 49, 737–759, 1987.

[2] Gh. Paun., G. Rozenberg, A. Salomaa, *Computing by splicing*, Theor. Comp. Science, 168, 321–336, 1996.

[3] Gh. Paun., G. Rozenberg, A. Salomaa, *DNA Computing: new computing paradigms*, Springer-Verlag, 1998.

[4] D. Pixton, *Regularity of splicing languages*, Discrete Appl. Math., 69, 101–124, 1996.