Separating some splicing models

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

Dipartimento di Informatica Sistemistica e Comunicazione
via Bicocca degli Arcimboldi 8, 20136 Milano
Università degli Studi di Milano-Bicocca - ITALY

contact: ferretti@disco.unimib.it

Abstract

This work shows that the family of languages generated by splicing as defined in the early paper by T.Head is strictly included in the family defined by later definitions by Head, Gh.Paun, and others. It also proves the strict inclusion of this latter family in the splicing family defined by D.Pixton when proving the regularity of splicing languages.

We describe instance languages in the difference sets, and prove how they cannot be generated by the smaller families.

1 Introduction

In 1987 Tom Head ([1]) defined splicing languages to be modeled on the combined action on DNA molecules of restriction enzymes and ligands. Enzymes are able to *cut* DNA molecules where they find specific patterns along the double helix. On the other hand, ligands are able to tie together molecules previously cut by restriction enzymes, but only when the two pieces can go together, that is, if the cut left single stranded some nucleotides then the corresponding nucleotides going to be pasted have to be complementary, in a well defined chemical sense.

Head translated this in the framework of formal languages, basing a splicing system on triples (α, μ, β) which define the patterns looked after by the restriction enzymes, and thus allowing the cut when they occurs. In particular, μ (the *crossing*) defines the identity of the group of nucleotides left single stranded: only strings cut according to triples having the same μ can be tied together.

When two strings, previously cut, are going to be tied, the pair of triples which originated those two strings are related *only* by the identical μ pattern. Crossings define an equivalence relation among the triples.

In 1992 Head ([2]) redefined his model of splicing, no more considering patterns defined by triples, and rules for pasting dictated by the μ patterns. He defined splicing systems where the patterns activating the cut are pairs (x, y). On these pairs a reflexive and symmetric relation T is defined, and the cut strings are allowed to be tied together only when they originate from cuts operated by two pattern pairs in relation according to T.

Finally, Paun (see for instance [3]) defined a slightly different formalism for splicing systems, where the patterns are again pairs, but the relation among them could be, generally, any binary relation.

This effectively linked 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 cut, the generated strings are immediately pasted together with no interference from strings cut from some other pattern: we just relate in T 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.

This was not possible in the first definition by Head, where the string μ in the pattern also directly created the relation allowing to paste the strings. For instance, one could not design two distinct groups of patterns, whose generated strings could not be cross-pasted, if all these patterns needed to have the identical empty μ. And this is quite an inconvenience, as we will see by giving a language which needs distinct groups of patterns with empty μ, and thus cannot be generated under the original definition of splicing.

One more different definition of splicing was given by Pixton ([4]), where the cutting step and the pasting step are again operated one immediately after the other, but one is even allowed to *substitute* part of the pattern before pasting the strings originated by the cut. And this also is far from what is possible in the other definitions of splicing, where instead the final string only could include substrings coming from one or the other of the originating strings. We will describe a language exploiting this further possibility, and which cannot be generated according to the other splicing formalisms.

2 Definitions

We recall shortly the main formal definitions from the papers [1], [3] and [4].

A *Head system* is defined as the tuple $S_H = (\Sigma, I, B, C)$, for $I \subset \Sigma^*$ a finite set of initial strings, B and C finite sets of triples, called *patterns* (α, μ, β), with $\alpha, \beta, \mu \in \Sigma^*$. The string μ of a triple is also called *crossing* of the triple. Given two words $u\alpha\mu\beta v, p\alpha'\mu\beta' q \in \Sigma^*$ 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 $p\alpha'\mu\beta v$. Then the *Head splicing language* generated by S_H is the $L = L(S_H)$ which is the smallest language obtained by using the splicing operation starting with I.

A *Paun system* is a tuple $S = (A, I, R)$, with A a finite alphabet, 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 words $w'u'u''w'', z'v'v''z'' \in A^*$, the rule $u'|u''\$v'|v''$ produces by splicing the words $w'u'v''z'', z'v'u''w''$. The *Paun splicing language* generated by S is the smallest language obtained by applying the rules in R starting with I.

A *Pixton system* consists of the pair (I, R), where I is a finite language, the initial set of axioms, and R is a finite set of rules, where each rule r is a triple

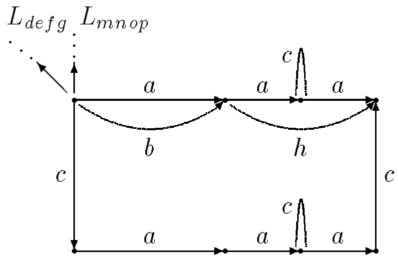

Figure 1: *A Paun splicing language which is not a Head splicing language.*

(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 $x_4 = v\beta w$. The *Pixton splicing language* generated by (I, R) is the smallest language obtained by applying the rules in R starting with I.

3 Differences between the models

First we examine a language in Paun splicing languages and not in Head splicing languages:

Example. (See Figure 1)
Given an alphabet containing, for instance, the letters a, b, c, h we define

$$L_{abch} = \{a^2c^*a, ca^2c^*ac, ba^*c, ah, bh\}.$$

Let L be the following regular language over alphabet $\Sigma_1 \cup \Sigma_2 \cup \Sigma_3$, for $\Sigma_1 = \{a, b, c, h\}, \Sigma_2 = \{d, e, f, g\}, \Sigma_3 = \{m, n, o, p\}$:

$$L = L_{abch} \cup L_{defg} \cup L_{mnop}.$$

□

Lemma 1 *The language L of the above example is generated by a Paun system and is not generated by a Head splicing system.*

Proof. We first give the Paun system which generates L. Since L is the union of three isomorphic languages, we first construct the Paun system S_P generating L_{abch}. Then $S_P = (\Sigma_1, I, R)$, for $I = \{a^2c^na,\ bh, bac^na,\ ca^2c^nac, ah\}$, where $0 \leq n \leq 2$, $R = \{ba|c\$bac|c, ca^2|c\$ca^2c|c, b|a\$a|h\}$. We leave to the reader the proof that S_P generates L_1. Similarly, we can define the Paun system generating the other two languages which build L. Since they all are over distinct alphabets, it is immediate to obtain the Paun system for generating L.

Let us now show that L is not generated by a Head system. The proof is by contradiction. Thus let us assume to the contrary that $L = L(S_H)$ for some system $S_H = (\Sigma, I, B, C)$.

Now, given a 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 $|\mu| < j$, it follows that $\mu = a^ic^k$ or $\mu = c^ka^t$ for $0 \leq i \leq 2$, $0 \leq k \leq j$ and $0 \leq t \leq 1$. Thus the following two cases must be considered.

Case i: Let μ contain a c letter. It follows that p_1 is a substring of a string $x = ca^2c^tac$; thus 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 a string ca^2c^tac, we obtain as in case i a contradiction. Thus, let us assume that p_1 is not a substring of ca^2c^*ac. Then it must be that $p_1 = (\epsilon, a, h)$. If $p_1 = (\epsilon, a, h)$, then $p_2 = (\delta, a, ac^ta^l)$, for $0 \leq l \leq 1$, $0 \leq t \leq j$. But, it is immediate to verify that p_1 and p_2 generate strings in a^2c^*ac which are not in L, a contradiction.

Case i and ii show that μ must be empty.

Similarly, as above, we can show that given an arbitrary string $z_j = d^2e^jd \in L$, with j arbitrarily large, then z_j is derived by using a pair of patterns p_3, p_4 whose crossing must be empty.

In 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 Σ_1 and Σ_2 in B, and Σ_3 in C. Consequently, for the strings considered above pattern p_1 can be used with pattern p_3 or p_4 to derive strings which contain letters from both alphabets Σ_1 and Σ_2, and hence which are not in L. □

Let us consider now the following language which is generated by a Pixton system but which is not generated by a Paun system.

Example. (See Figure 2)
Consider the language

$$L = \{cx^*ae, cx^*be, dcx^*bef\}.$$

□

Lemma 2 *The language L of the above example is generated by a Pixton system and is not generated by a Paun splicing system.*

Proof. The language L can be generated by a Pixton system. In fact, let $I = \{cae, cxae, dcbef, dcxbef\}$, and $R = \{r_1 = (a, xa, xa), r_2 = (dcx, dcx, dc), r_3 =$

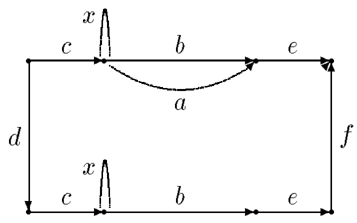

Figure 2: *A Pixton splicing language which is not a Paun splicing language.*

$(a, b, a)\}$. Then, by the first rule r_1 applied to 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 $\{dcx^*xbef\}$. 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.

On the other hand, the language L is not generated by Paun systems. Any such 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 chose those substring and avoid the cross-over between strings of the form cx^*be and strings of the form 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 systems. □

4 Further observations

Clearly, given their respective definitions, Head splicing languages are always Paun splicing languages, and these are always Pixton splicing languages. Thus we would prove the strict inclusion between these language classes.

We observe again that it can be useful to compare these different models of splicing languages by underlining the differences with respect to the binary relation defined on the patterns which specify where to cut. In [1] that relation was in fact a transitive relation, being based in the identity of the crossing in the patterns. In [2] it was required to be reflexive and symmetric. In [3] it could be *any* relation, in our understanding.

For instance, it can be understood how the following result was proved in [1] and directly inherited in the different formalism of [2]: a language is strictly locally testable if and only if it is generated by a *null context* splicing system. In the first formalism null context meant patterns of the form $(\epsilon, \mu, \epsilon)$, with the *implicit* consequence that they created an identity relation. In the second formalism null context was defined as requiring patterns of the form (μ, ϵ) and *explicitly* requiring an identity relation among them.

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] T. Head. Splicing schemes and DNA. *Lindenmayer Systems: Impacts on Theoretical Computer Science and Developmental Biology*, Springer-Verlag, Berlin, 371–383, 1992.

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

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