ELSEVIER
Theoretical Computer Science 210 (1999) 227–243
Theoretical Computer Science

Splicing on tree-like structures

Yasubumi Sakakibara [a,*], Claudio Ferretti [b]

[a] *Department of Information Sciences, Tokyo Denki University, Hatoyama, Hiki-gun, Saitama 350-03, Japan*

[b] *Dipartimento di Scienze dell'Informazione, Università di Milano via Comelico 39, 20135 Milano, Italy*

Abstract

In this paper, we provide a method to increase the power of splicing systems. We introduce the splicing systems on trees to be built as partially annealed single strands, which is a quite similar notion and a natural extension of splicing systems on strings. Trees are a common and useful data structure in computer science and have a biological counterpart such as molecular sequences with secondary structures, which are typical structures in RNA sequences. Splicing on trees involves (1) complete subtrees as axioms, (2) restriction operated on the annealed subsequences, (3) rules to substitute a complete subtree with another. We show that splicing systems on trees with finite sets of axioms and finite sets of rules can generate the class of context-free languages without the need of imposing multiplicity constraints.

Keywords: Splicing system; DNA computer; Context-free grammer; Tree

1. Introduction

We extend the notion of splicing systems, usually defined on strings, to splicing systems on trees.

Splicing systems (also called H systems) have been developed as a new computational tool for generating formal languages, and are inspired by molecular biology [3]. Splicing operations were originally defined as those that work on linear strings like DNA sequences. Main theoretical result which has been established in splicing systems is that extended H systems with finite sets of axioms and finite sets of rules can exactly generate the class of regular languages [6]. They can characterize the class of recursively enumerable languages, if we allow constraints on the number of copies of some molecules in the system, instead of assuming that they all have infinitely many

* Corresponding author. E-mail: yasu@j.dendai.ac.jp.

0304-3975/99/$ – see front matter
PII S0304-3975(98)00087-5

duplicates [2]. However, such a multiplicity constraint is very unnatural from the biological point of view, and further it destroys the advantage of massive parallelism in molecular computation [1].

In this paper, we provide a method to increase the power of splicing systems. We introduce the splicing systems on trees, to be built as partially annealed single strands, and this is a notion quite similar to that of splicing systems on strings, and a natural extension of it. Trees are a common and useful data structure in computer science and have a biological counterpart such as molecular sequences with secondary structures, which are typical structures in RNA sequences. This extension is motivated by ideas from two fields of research: genetic programming [4] and stochastic context-free grammars for modeling RNA [8].

Genetic programming is the extension of the genetic algorithm into the space of tree-like structures. That is, the objects that constitute the population are not fixed-length character strings that encode possible solutions, they are programs expressed in tree-like structures. This extension succeeds to provide the substantial increase of the power and applicability of genetic algorithm.

Stochastic context-free grammars have successfully been applied to modeling families of RNA sequences and their effectiveness has been shown in computational biology problems such as folding, aligning and discriminating RNA sequences. This research seems to imply the possibility of splicing on secondary structures which are typical in RNA families to get a generative power for context-free languages.

1.1. Introduction to splicing

Splicing is a model of the recombinant behavior of double stranded molecules of DNA under the action of restriction enzymes and lygases. A single strand of DNA is an oriented sequence of nucleotides A, C, G and T, but since A can bind to T and G to C, two strands of DNA bind together to form a doubled stranded DNA molecule, if they have matching pairs of nucleotides when reading the second one along the reverse orientation. For instance, the strands AAGC and GCTT bind together to form a double stranded DNA molecule. Of course, given the fixed pairings between nucleotides, we can model such a molecule simply as *aagc*, moving toward the realm of formal languages.

Restriction enzymes operate on double stranded DNA molecules recognizing specific subsequences of nucleotides, and *cut* each strand at fixed points inside that matching site, generally leaving short single stranded overhangs. We can represent the action of a restriction enzyme on a string as a triple (a,x,b), where axb is the recognized site, and x is the central single stranded segment between the two cuts operated on each strand.

Lygases are able to join together two complementary single stranded overhangs, always according to the A-T and G-C pairings of nucleotides. This means that two strings *saxbt* and *pcxdq*, first cut by two restriction enzymes (a,x,b) and (c,x,d), can

either recombine in the original form or they can build two strings *saxdq* and *pcxbt*, since lygases can operate on the common complementary overhang x.

In this way we can model a test tube, containing DNA molecules together with a set of restriction enzymes and lygases, as a generating mechanism of formal languages: the starting set of strings is continuously cut and pasted to generate new strings. Then we can rise a theoretical question: which language will be generated?

In the scientific literature, characterizations in terms of splicing systems have already been given for language classes such as locally testable languages, regular languages, and recursively enumerable languages. These results are obtained using variations on the definition of the splicing systems. In all these results restriction enzymes are modeled in pairs described by quadruples, instead than by a triple for each enzyme as we have seen above. A pair of restriction enzymes is represented by a *rule* of the form:

$$r = u_1 \# u_2 \$ u_3 \# u_4$$

Such a rule directly transforms the strings su_1u_2t and pu_3u_4q into su_1u_4q and pu_3u_2q, applying cut and paste between them. Making further assumptions on the behavior of the splicing system leads to the characterization of various language classes:

- the class of locally testable languages can be characterized by splicing systems where the splicing of two strings on a restriction site containing as a substring another site, does not destroy the latter (*persistent* systems [3]);
- a way to generate all and only the regular languages is to use splicing systems where, among all the generated strings, we can select only those without some special symbols (*extended* splicing systems [2]);
- splicing systems where most molecules are present in an unlimited number of copies, but where some strings have only a finite number of duplicates, characterize the class of recursively enumerable languages (*multiplicity constraints* [2]).

In this paper we define another variation of the splicing operation, with the goal of obtaining results concerning context free languages. This class is strangely almost absent in the literature of this field. For instance, in [6] there is a characterization of context free languages, provided that we can use any language from that same class as starting set of strings for the splicing system. But when we consider to use only finite sets of axioms, as it is also done in the present paper, known characterizations go suddenly from the class of regular language to the universal class.

2. Preliminaries

A *tree* over an alphabet V is a rooted, directed, connected acyclic finite graph in which the direct successors of any node are linearly-ordered from left to right. The formal definition of the tree is as follows. Let $\mathbb{N}$ be the set of natural numbers. A *ranked alphabet* V is a finite set of symbols associated with a finite relation called the

rank relation $r_V \subseteq V \times \mathbb{N}$. V_n denotes the subset $\{f \in V \mid (f,n) \in r_V\}$ of V. In many cases the symbols in V_n are considered as *function symbols*. We say that a function symbol f has an *arity* n if $f \in V_n$ and a symbol of arity 0 is called a *constant symbol* which corresponds to a terminal symbol in the grammar.

A *tree* over V is a mapping t from Dom_t into V where (1) the domain Dom_t is a finite nonempty subset of $\mathbb{N}^*$; (2) if $x \in Dom_t$ and $y < x$, then $y \in Dom_t$; (3) if $y \cdot i \in Dom_t$ and $i \in \mathbb{N}$, then $y \cdot j \in Dom_t$ for $0 \leqslant j \leqslant i$, $j \in \mathbb{N}$; (4) $t(x) \in V_n$, whenever for $i \in \mathbb{N}$, $x \cdot i \in Dom_t$ if and only if $0 \leqslant i \leqslant n-1$. An element of the tree domain Dom_t is called a *node* of t. If $t(x) = A$, then we say that A is the *label* of the node x of t. If we consider V as a set of function symbols, the trees over V can be identified with *well-formed terms* over V and written linearly with commas and parentheses. Let V^T denote the set of all trees over V. Let λ denote the null tree in V^T.

For the notational convention, we often use the form $u(u_1,\ldots,u_n)$ to denote the tree defined by

$$u(u_1,\ldots,u_n)\ (x) = \begin{cases} u(x) & \text{if } x \in Dom_u \text{ and } u(x) \neq \lambda_i, \\ u_i(y) & \text{if } x = z \cdot y,\ u(z) = \lambda_i \text{ and } y \in Dom_{u_i} \\ & \quad \text{for } 1 \leqslant i \leqslant n \end{cases}$$

for the tree $u(\lambda_1,\ldots,\lambda_n) \in (V \cup \{\lambda\})^T$ (where λ_i denotes the null tree) and $u_i \in V^T$ $(1 \leqslant i \leqslant n)$. That is, $u(u_1,\ldots,u_n)$ is the tree obtained from $u(\lambda_1,\ldots,\lambda_n)$ by replacing each null tree λ_i with a tree u_i.

Formally, a *grammar* is composed of three parts. The first is a finite alphabet Σ of *terminal symbols*. For DNA sequences, this alphabet comprises the four nucleotides `A`, `C`, `G` and `T`, and for RNA sequences, the nucleotides `A`, `C`, `G` and `U` instead of `T`. The second is a finite set N of *nonterminal symbols* and a special *start symbol* S. The third is a set P of rewrite rules (or *production rules*) that specify how sequences containing nonterminals may be rewritten by expanding those embedded nonterminals to new subsequences. The language generated by the grammar is the set of all sequences of terminal symbols that can be derived from the start symbol S by repeatedly applying productions from P.

A *context-free grammar* (CFG) is defined by a quadruple $G = (N,\Sigma,P,S)$. Each production rule in P has the form $X \to \alpha$ for $X \in N$ and $\alpha \in (N \cup \Sigma)^*$, indicating that the nonterminal X can be replaced by the sequence α. The *language generated* by a CFG G is denoted $L(G)$.

A *derivation* is a rewriting of a sequence using the rules of the grammar. It begins with a sequence that consists only of the start symbol S. In each step of the derivation, a nonterminal from the current sequence is chosen and replaced with the right-hand side of a production rule for that nonterminal. This replacement process is repeated until the sequence consists of terminal symbols only. A simple derivation is shown in Fig. 1.

A derivation can be arranged in a tree structure called a *derivation tree*. Let $D(G)$ denote the set of derivation trees of a CFG G.

a. Production rules

$$
\begin{aligned}
P = \{\ & S_0 \to S_1, && S_7 \to \mathtt{G}\ S_8,\\
& S_1 \to \mathtt{C}\ S_2\ \mathtt{G}, && S_8 \to \mathtt{G},\\
& S_1 \to \mathtt{A}\ S_2\ \mathtt{U}, && S_8 \to \mathtt{U},\\
& S_2 \to \mathtt{A}\ S_3\ \mathtt{U}, && S_9 \to \mathtt{A}\ S_{10}\ \mathtt{U},\\
& S_3 \to S_4\ S_9, && S_{10} \to \mathtt{C}\ S_{10}\ \mathtt{G},\\
& S_4 \to \mathtt{U}\ S_5\ \mathtt{A}, && S_{10} \to \mathtt{G}\ S_{11}\ \mathtt{C},\\
& S_5 \to \mathtt{C}\ S_6\ \mathtt{G}, && S_{11} \to \mathtt{A}\ S_{12}\ \mathtt{U},\\
& S_6 \to \mathtt{A}\ S_7, && S_{12} \to \mathtt{U}\ S_{13},\\
& S_7 \to \mathtt{U}\ S_7, && S_{13} \to \mathtt{C} \qquad \}
\end{aligned}
$$

b. Derivation

$$
\begin{aligned}
S_0 &\Rightarrow S_1 \Rightarrow \mathtt{C}S_2\mathtt{G} \Rightarrow \mathtt{CA}S_3\mathtt{UG} \Rightarrow \mathtt{CA}S_4S_9\mathtt{UG}\\
&\Rightarrow \mathtt{CAU}S_5\mathtt{A}S_9\mathtt{UG} \Rightarrow \mathtt{CAUC}S_6\mathtt{GA}S_9\mathtt{UG}\\
&\Rightarrow \mathtt{CAUCA}S_7\mathtt{GA}S_9\mathtt{UG} \Rightarrow \mathtt{CAUCAG}S_8\mathtt{GA}S_9\mathtt{UG}\\
&\Rightarrow \mathtt{CAUCAGGGA}S_9\mathtt{UG} \Rightarrow \mathtt{CAUCAGGGAA}S_{10}\mathtt{UUG}\\
&\Rightarrow \mathtt{CAUCAGGGAAG}S_{11}\mathtt{CUUG}\\
&\Rightarrow \mathtt{CAUCAGGGAAGA}S_{12}\mathtt{UCUUG}\\
&\Rightarrow \mathtt{CAUCAGGGAAGAU}S_{13}\mathtt{UCUUG}\\
&\Rightarrow \mathtt{CAUCAGGGAAGAUCUCUUG}.
\end{aligned}
$$

c. Parse tree

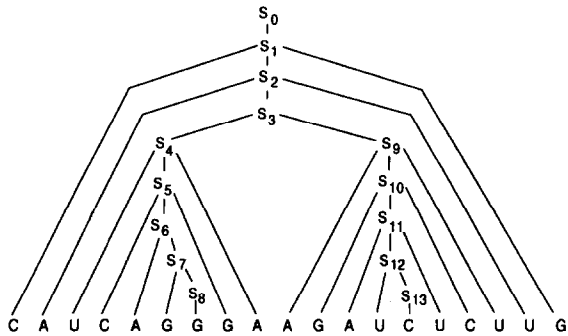


Fig. 1. (a) A simple CFG which may be used to derive a set of RNA sequences including the specific example illustrated here, CAUCAGGGAAGAUCUCUUG. S_0 (start symbol), $S_1,\ldots,S_{13}$ are nonterminal symbols; A, U, G and C are terminal symbols representing the four nucleotides. (b) Application of the production rules P could generate the example sequence by the derivation indicated. (c) The derivation process in b may be arranged in a tree structure called a *derivation tree*.

2.1. Splicing systems on tree structures

In the following definitions we basically move from splicing on linear strings, to splicing on tree-like structures that still resemble some natural molecular assemblies. The same generative mechanism, based on starting structures and rules operating by cut and paste on them, will be kept. We use finite set of splicing rules described by quadruples, instead of the triples of [3], but we still use finite set of starting strings and of rules. We do not make assumptions such as that of having persistent splicing systems, and we neither impose multiplicity constraints.

We define the splicing operation on tree structures. A *splicing rule on tree structures* over V is of the form

$$r = u_1(\lambda,\ldots,\#(u_2(\lambda,\ldots,\lambda)),\ldots,\lambda)\ \$\ u_3(\lambda,\ldots,\#(u_4(\lambda,\ldots,\lambda)),\ldots,\lambda)$$

for $u_i \in V^T$ $(1 \leqslant i \leqslant 4)$ and for two special symbols #, \$ not in V. # is used as a marker for specifying a single node in a tree. (See Fig. 2.)

For such a rule r, suppose two trees x and y in V^T which contain $u_1(\lambda,\ldots,u_2(\lambda,\ldots,\lambda),\ldots,\lambda)$ and $u_3(\lambda,\ldots,u_4(\lambda,\ldots,\lambda),\ldots,\lambda)$ as a part of the tree respectively. That is, x con-

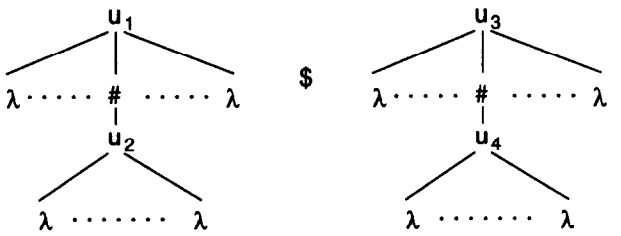


Fig. 2. Splicing rule on tree structures.

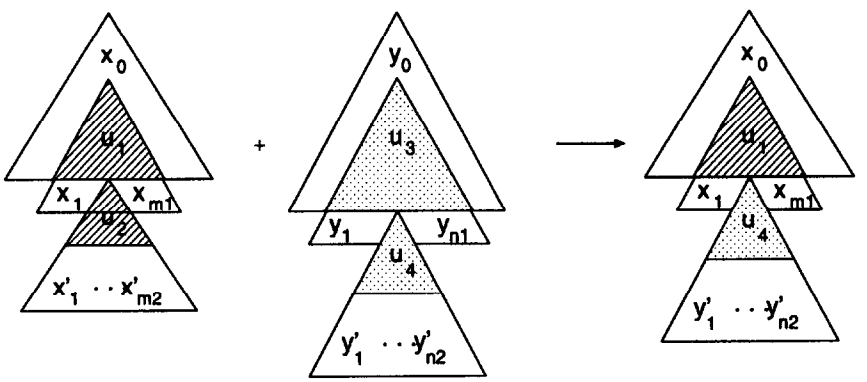


Fig. 3. Splicing on tree structures.

tains a subtree constructed from $u_1(\lambda,\ldots,u_2(\lambda,\ldots,\lambda),\ldots,\lambda)$ by replacing each null tree λ by some tree in V^T. For such a rule r, such trees $x, y \in V^T$, and a tree $z \in V^T$, we define the operation $\vdash_r$ as follows:

$$
\begin{aligned}
(x,y) \vdash_r z \text{ iff } & x = x_0(u_1(x_1,\ldots,u_2(x'_1,\ldots,x'_{m_2}),\ldots,x_{m_1})) \\
& y = y_0(u_3(y_1,\ldots,u_4(y'_1,\ldots,y'_{n_2}),\ldots,y_{n_1})) \\
& z = x_0(u_1(x_1,\ldots,u_4(y'_1,\ldots,y'_{n_2}),\ldots,x_{m_1})) \\
& \text{for some } x_0,\ldots,x_{m_1},x'_1,\ldots,x'_{m_2},y_0,\ldots,y_{n_1},y'_1,\ldots,y'_{n_2} \in V^T.
\end{aligned}
$$

(See Fig. 3.)

A *splicing scheme on tree structures* is a pair $\gamma = (V,R)$, where V is an alphabet and R is a set of splicing rules on tree structures over V. A pair $\mathscr{S} = (\gamma,L)$ for a splicing scheme γ and a tree language $L \subseteq V^T$ is called *splicing system on tree structures* (or HT system).

For an HT system $\mathscr{S} = (\gamma,L)$, we define

$$\gamma(L) = \{z \in V^T \mid (x,y) \vdash_r z \text{ for some } x,y \in L, r \in R\}.$$

We also define the *iterated splicing* as follows:

$$\gamma^0(L) = L,$$
$$\gamma^{i+1}(L) = \gamma^i(L) \cup \gamma(\gamma^i(L)), \quad i \geqslant 0,$$
$$\gamma^*(L) = \bigcup_{i \geqslant 0} \gamma^i(L).$$

For a class of tree languages F_1 and a class of sets of splicing rules F_2, we denote by $HT(F_1, F_2)$ the set of all tree languages $\gamma^*(L)$, for $L \in F_1$ and $\gamma = (V, R)$, $R \in F_2$. Especially, we denote by FIN_T the class of finite tree languages, by FIN_R the class of finite sets of splicing rules, and by $HT(FIN_T, FIN_R)$ the set of all tree languages $\gamma^*(L)$ generated from a finite tree language $L \in FIN_T$ and a finite set of splicing rules on tree structures in FIN_R.

3. RNA secondary structures and tree structures

We demonstrate that a special kind of trees, called skeletons, have a biological counterpart such as RNA sequences with secondary structures and splicing operations on skeletons correspond to splicing on secondary structures of molecular sequences.

Let σ denote a special symbol used for labeling internal nodes of trees. A tree defined over $(\{\sigma\} \cup \Sigma)$ is called a *skeleton*. A skeleton is a tree in which all the internal nodes are labeled by σ and other nodes are by symbols in Σ. A skeleton describes only the shape and terminal nodes of the derivation tree.

On the other hand, in RNA, the nucleotides A, C, G and U interact in specific ways to form characteristic secondary-structure motifs such as helices, loops and bulges. In general, the folding of an RNA chain into a functional molecule is largely governed by the formation of intramolecular A-U and G-C Watson–Crick pairs. Such base pairs constitute the so-called *biological palindromes* in the genome.

We consider that such a physical secondary structure of molecular sequence is a reflection of a skeleton. A skeleton represents the syntactic structure of an RNA sequence (Fig. 4, right), and this syntactic structure corresponds to the physical secondary structure (Fig. 4, left). Therefore, splicing on secondary structures of RNA sequences (like shown in Fig. 5) corresponds to splicing on skeletons. We call the HT system on skeletons the *HTS system*. Thus HTS system is a formalization for splicing on secondary structures.

4. The power of HT systems

First we show that splicing on tree structures generates exactly the class of context-free languages. Next we show our main result that splicing on skeletons is almost equal to the power of context-free grammars. Both proofs are natural extensions of the one (Theorem 9) in Paun [5].

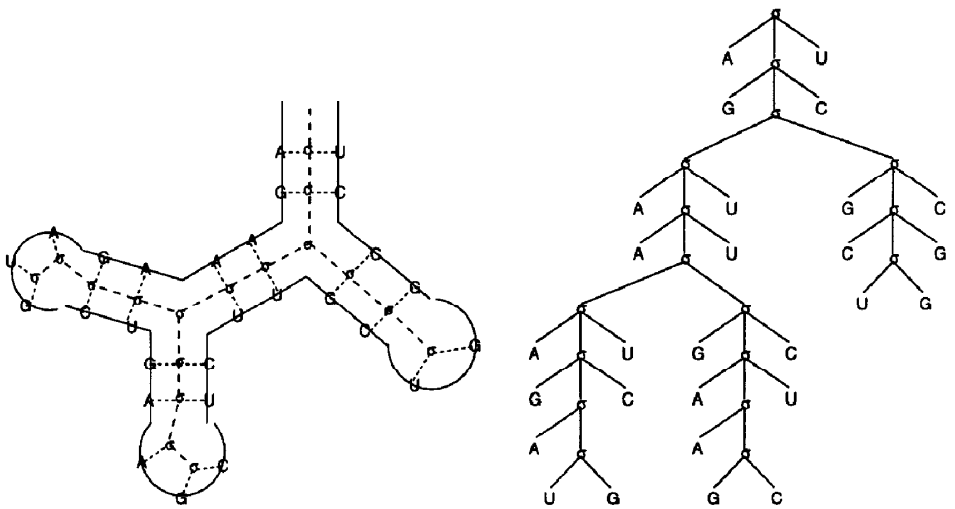


Fig. 4. A physical secondary structure of RNA sequence and a derivation tree (skeleton) which corresponds to the secondary structure.

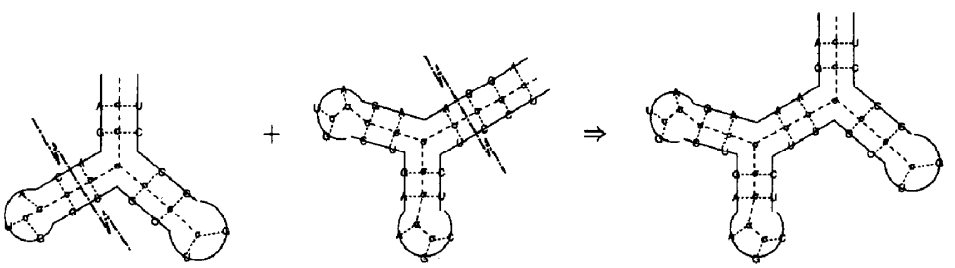


Fig. 5. Splicing operation on secondary structures.

Theorem 1. *For any context-free grammars G, the set of derivation trees of G is a tree language in the class $HT(FIN_T, FIN_R)$ over $V = N \cup \Sigma$.*

Proof. Let $D(G)$ be the set of derivation trees generated by a CFG $G = (N, \Sigma, P, S)$ with the rules in P of the forms $X \to YZ$, $X \to aY$, and $X \to a$, for $X, Y, Z \in N$, $a \in \Sigma$ (it is clear that these rules are enough for generating every CFG because it includes the Chomsky normal form).

Consider the alphabet

$$V = N \cup \Sigma,$$

the splicing scheme

$$\begin{aligned}\gamma = (V,\ & \{X(\#(Y),Z)\ \$\ \lambda(\#(Y)) \mid X,Y,Z \in N\} \cup \\ & \{X(Y,\#(Z))\ \$\ \lambda(\#(Z)) \mid X,Y,Z \in N\} \cup \\ & \{X(a,\#(Y))\ \$\ \lambda(\#(Y)) \mid X,Y \in N, a \in \Sigma\}),\end{aligned}$$

and the finite tree language $L_0 \subseteq D(G)$ defined by

$$L_0 = \{t \in D(G) \mid \text{every path in } t \text{ has at most two occurrences of the same rule in } P\}.$$

We show the relation

$$D(G) = \gamma^*(L_0).$$

First, each tree in L_0 is obviously a derivation tree of G. Next, if x and y are trees in $\gamma^i(L_0)$ and derivation trees of G, and of the forms $x_1(X(U(x_2), Y(x_3)))$ and $y_1(Y(y_2))$ respectively, then $z = x_1(X(U(x_2), Y(y_2))) \in \gamma^{i+1}(L_0)$ and obviously z corresponds to a derivation tree of G, too. Hence $\gamma^*(L_0) \subseteq D(G)$.

Conversely, consider the derivation trees of G. Such derivation trees t of depth less than two are in L_0, hence in $\gamma^*(L_0)$. Assume that all such derivation trees of depth less than some $n \geqslant 2$ are in $\gamma^*(L_0)$ and consider a derivation tree t of the smallest depth greater than n in $D(G)$. If $t \notin L_0$, it follows that a path in t contains a production rule $X \to UY$ with $X, Y \in N$ and $U \in N \cup \Sigma$ on three different positions:

$$t = x_0(\, X(U(u_1), Y(x_1(\, X(U(u_2), Y(x_2(\, X(U(u_3), Y(x_3))) \,)) \,)) \,).$$

Then

$$t' = x_0(X(U(u_1), Y(x_1(X(U(u_2), Y(x_3)))))),$$

$$t'' = x_0(X(U(u_1), Y(x_2(X(U(u_3), Y(x_3)))))).$$

are correct derivation trees of G and the depths of both trees t' and t'' are less than the depth of t and hence less than n. Therefore $t', t'' \in \gamma^*(L_0)$. From the form of t', t'' and of splicing rules of γ, it is obvious that $t \in \gamma^*(L_0)$. Hence $D(G) \subseteq \gamma^*(L_0)$. □

Now we show that the class HTS (FIN_T, FIN_R) is almost equal to the class of derivation trees of context-free grammars. We consider a mapping g from skeletons to derivation trees. For a set of skeletons L, let $g(L)$ denote the set of derivation trees $\{g(s) \mid s \in L\}$ and we call it the *coding* of L.

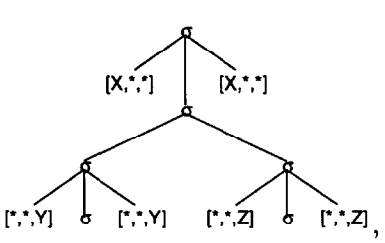
Theorem 2. *For a mapping g from skeletons to derivation trees, the set of derivation trees of a context-free grammars G is the coding of a tree language in the class* $HTS(FIN_T, FIN_R)$.

Proof. Let $D(G)$ be the set of derivation trees generated by a CFG $G = (N, \Sigma, P, S)$ with the rules in P of the forms $X \to YZ$, $X \to aY$, and $X \to a$, for $X, Y, Z \in N$, $a \in \Sigma$.

Consider the alphabet

$$V = \{\sigma([X, *, *],\ \sigma(\sigma([*, *, Y], \lambda, [*, *, Y]), \sigma([*, *, Z], \lambda, [*, *, Z])),\ [X, *, *])$$
$$\mid X \to YZ \in P,\ X, Y, Z \in N, \text{a special symbol } * \text{ not in } N,\ \Sigma\}$$

$X \to YZ,$



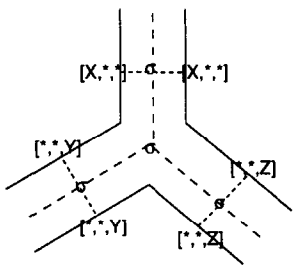


Fig. 6. A production rule of the form $X \to YZ$, its tree representation on skeletons, and the molecular implementation.

$$\cup\{\sigma([X,a,Y],\lambda) \mid X \to aY \in P,\ X,Y \in N, a \in \Sigma\}$$

$$\cup\{\sigma([X,a,*]) \mid X \to a \in P,\ X \in N, a \in \Sigma\}.$$

The symbols in the alphabet V represent the production rules in P in the forms of skeletons (see Fig. 6). Note that any tree over V is a skeleton over $\{\sigma\} \cup \Sigma'$ for $\Sigma' = \{[X,*,*],[*,*,X],[X,a,Y],[X,a,*] \mid X,Y \in N, a \in \Sigma\}$. Consider also the splicing scheme

$$\begin{aligned}
\gamma = (V,\ & \{\sigma([*,*,X],\#(\lambda),[*,*,X])\ \$\ \lambda(\#(\sigma([X,*,*],\lambda,[X,*,*]))) \mid X \in N\} \\
& \cup \{\sigma([*,*,X],\#(\lambda),[*,*,X])\ \$\ \lambda(\#(\sigma([X,a,Y],\lambda))) \mid X,Y \in N, a \in \Sigma\} \\
& \cup \{\sigma([*,*,X],\#(\lambda),[*,*,X])\ \$\ \lambda(\#(\sigma([X,a,*]))) \mid X \in N, a \in \Sigma\} \\
& \cup \{\sigma([X,a,Y],\#(\lambda))\ \$\ \lambda(\#(\sigma([Y,*,*],\lambda,[Y,*,*]))) \mid X,Y \in N, a \in \Sigma\} \\
& \cup \{\sigma([X,a,Y],\#(\lambda))\ \$\ \lambda(\#(\sigma([Y,b,Z],\lambda))) \mid X,Y \in N, a,b \in \Sigma\} \\
& \cup \{\sigma([X,a,Y],\#(\lambda))\ \$\ \lambda(\#(\sigma([Y,b,*]))) \mid X,Y \in N, a,b \in \Sigma\}),
\end{aligned}$$

and the finite tree language L_0 defined by

$$L_0 = g(L_0'),$$

where

$$L_0' = \{t \in D(G) \mid \text{every path in } t \text{ has at most two occurrences of the same rule in } P\},$$

and g is a mapping from $(N \cup \Sigma)^T$ to V^T defined recursively by

$$\begin{aligned}
g(X(Y(t_1),Z(t_2))) = \sigma([X,*,*],\sigma(\sigma([*,*,Y],g(Y(t_1))),[*,*,Y]), \\
\sigma([*,*,Z],g(Z(t_2))),[*,*,Z])),[X,*,*]),
\end{aligned}$$

$$g(X(a,Y(t_1))) = \sigma([X,a,Y],g(Y(t_1))),$$

$$g(X(a)) = \sigma([X,a,*]).$$

Note that $g(L_0')$ is a set of skeletons.

Consider also the mapping $h : V^T \to (N \cup \Sigma)^T$ defined recursively by

$$h(\sigma([X,*,*],\sigma(\sigma([*,*,Y],t_1,[*,*,Y]),\sigma([*,*,Z],t_2,[*,*,Z])),[X,*,*])) = X(h(t_1),h(t_2)),$$

$$h(\sigma([X,a,Y],t_1)) = X(a,h(t_1)),$$

$$h(\sigma([X,a,*])) = X(a).$$

Obviously, a pair $\mathscr{S} = (\gamma, L_0)$ is an HTS system. We show the relation

$$D(G) = h(\gamma^*(L_0)).$$

First, from the definitions of the mappings g and h, it is obvious that each skeleton in L_0 describes a derivation tree of G, that is, $h(g(t)) \in D(G)$ for $t \in L_0'$.

Next, assume that x and y are trees in $\gamma^i(L_0)$ describing derivation trees of G. That is, $h(x), h(y) \in D(G)$. If x and y are of the forms:

$$x_1(\sigma([*,*,X],x_2,[*,*,X])) \quad \text{and} \quad y_1(\sigma([X,*,*],y_2,[X,*,*]))$$

respectively, then

$$z = x_1(\sigma([*,*,X],\sigma([X,*,*],y_2,[X,*,*]),[*,*,X])) \in \gamma^{i+1}(L_0)$$

and obviously z describes a derivation tree of G, too. That is, $h(z) \in D(G)$. If x and y are of the forms $x_1(\sigma([*,*,X],x_2,[*,*,X]))$ and $y_1(\sigma([X,a,Y],y_2))$, respectively, then $z = x_1(\sigma([*,*,X], \sigma([X,a,Y],y_2),[*,*,X])) \in \gamma^{i+1}(L_0)$ and $h(z) \in D(G)$. If x and y are of the forms $x_1(\sigma([X,a,Y],x_2))$ and $y_1(\sigma([Y,*,*],y_2,[Y,*,*]))$, respectively, then $z = x_1(\sigma([X,a,Y],\sigma([Y,*,*],y_2,[Y,*,*]))) \in \gamma^{i+1}(L_0)$ and $h(z) \in D(G)$. If x and y are of the forms $x_1(\sigma([X,a,Y],x_2))$ and $y_1(\sigma([Y,b,Z],y_2))$, respectively, then $z = x_1(\sigma([X,a,Y],\sigma([Y,b,Z],\ y_2))) \in \gamma^{i+1}(L_0)$ and $h(z) \in D(G)$. Consequently, $h(\gamma^*(L_0)) \subseteq D(G)$.

Conversely, consider the skeletons s over V describing derivation trees of G. That is, $h(s) \in D(G)$. Such skeletons s which describe derivation trees of G of depth less than two are in L_0, hence in $\gamma^*(L_0)$. Assume that all such skeletons which describe derivation trees of G of depth less than some $n \geqslant 2$ are in $\gamma^*(L_0)$ and consider a skeleton s which describe a derivation tree of G of the smallest depth greater than n, that is, $h(s)$ is of the smallest depth greater than n in $D(G)$. If $s \notin L_0$, it follows that a path in s contains a symbol α in V on three different positions. If α is of the form $\alpha(\lambda,\lambda) = \sigma([X,*,*],\sigma(\sigma([*,*,Y],\lambda,[*,*,Y]),\sigma([*,*,Z],\lambda,[*,*,Z])),[X,*,*])$, assume

$$s = x_0(\alpha(y_1,x_1(\alpha(y_2,x_2(\alpha(y_3,x_3)))))).$$

Then

$$s' = x_0(\alpha(y_1,x_1(\alpha(y_2,x_3)))),$$

$$s'' = x_0(\alpha(y_1,x_2(\alpha(y_3,x_3)))).$$

describe correct derivation trees of G, that is, $h(s'), h(s'') \in D(G)$, and the depths of both trees s' and s'' are less than the depth of s. Hence the depths of $h(s')$ and $h(s'')$

are less than n. Therefore $s', s'' \in \gamma^*(L_0)$. From the form of s', s'' and of splicing rules of γ, it is obvious that $s \in \gamma^*(L_0)$. If α is of the form $\alpha(\lambda) = \sigma([X, a, Y], \lambda)$, assume

$$s = x_0(\alpha(x_1(\alpha(x_2(\alpha(x_3)))))).$$

Then

$$s' = x_0(\alpha(x_1(\alpha(x_3)))), \quad s'' = x_0(\alpha(x_2(\alpha(x_3)))).$$

describe correct derivation trees of G and the depths of both trees s' and s'' are less than the depth of s. Hence the depths of $h(s')$ and $h(s'')$ are less than n. Therefore $s', s'' \in \gamma^*(L_0)$. From the form of s', s'' and of splicing rules of γ, it is obvious that $s \in \gamma^*(L_0)$.

Consequently, for each derivation tree t of G, we find a skeleton $s \in \gamma^*(L_0)$ such that $h(s)$ is exactly the derivation tree t, i.e., $h(s) = t$. Hence $D(G) \subseteq h(\gamma^*(L_0))$. □

Conversely, we show that the generative power of splicing on tree structures is not greater than that of context-free grammars. That is, a tree language generated by splicing systems with an initial set of derivation trees is a set of derivation trees generated by a context-free grammar.

Theorem 3. *Let RT be the class of skeleton languages generated by context-free grammars. A tree language in the class $HTS(RT, FIN_R)$ is a set of skeletons generated by a context-free grammar G.*

Proof. The proof is based on Pixton's proof [7] for showing the regularity of linear splicing systems and uses the same type of induction on a non-linear notion of *complexity* for derivation trees (we will explain it later). His result is shown for splicing rules of the form $r = (\alpha, \alpha'; \beta)$ and is easily applied to our type of splicing rules using the form $(u_1u_2, u_3u_4; u_1u_4)$.

Let us define the mapping s from trees to skeletons. For a tree t,

$$s(t)(x) = \begin{cases} t(x) & \text{if } x \text{ is a terminal node,} \\ \sigma & \text{if } x \text{ is an internal node.} \end{cases}$$

Thus the skeleton tells us about the shape and terminal nodes of the tree. For a set of trees T, let us define $s(T) = \{s(t) \mid t \in T\}$.

Let $\gamma = (V, R)$ be a splicing scheme on skeletons for $R \in FIN_R$ and $V = (\{\sigma\} \cup \Sigma)$, and $L_0 \in RT$ be an initial skeleton language.

We first construct a set of production rules to capture the splicing scheme. For each splicing rule

$$r = u_1(\lambda, \ldots, \#(u_2(\lambda, \ldots, \lambda)), \ldots, \lambda) \$ u_3(\lambda, \ldots, \#(u_4(\lambda, \ldots, \lambda)), \ldots, \lambda)$$

in R, we construct a CFG $B^{(r)} = (N^{(r)}, \Sigma, P^{(r)}, S^{(r)})$ which only generates the skeleton

$$u_1(\lambda_1, \ldots, u_4(\lambda'_1, \ldots, \lambda'_n), \ldots, \lambda_m),$$

i.e.,

$$D(B^{(r)}) = \{b_1^{(r)}(\lambda_1, \ldots, b_4^{(r)}(\lambda'_1, \ldots, \lambda'_n), \ldots, \lambda_m)\},$$

$$s(b_1^{(r)}(b_4^{(r)})) = u_1(u_4),$$

where λ_i and λ'_j denote null trees and we denote by $x(y)$ the tree $x(\lambda, \ldots, y(\lambda, \ldots, \lambda), \ldots, \lambda)$ for short. We refer to the production rule of the form $A \to \lambda_i$ in $P^{(r)}$ as $Z_i^{(r)} \to \lambda_i$ for $1 \leqslant i \leqslant m$ and the production rule of the form $A \to \lambda'_j$ as $\hat{Z}_j^{(r)} \to \lambda'_j$ for $1 \leqslant j \leqslant n$. We construct these CFGs $B^{(r)}$ $(r = 1, \ldots, |R|)$ to be pairwise disjoint and we let B be their union $B^{(1)} \cup B^{(2)} \cup \cdots \cup B^{(R)}$.

Let $G_0 = (N_0, \Sigma, P_0, S)$ be a CFG which generates L_0, that is, $s(D(G_0)) = L_0$. We assume G_0 is disjoint from B. We will construct a sequence of CFGs $G_k = (N_k, \Sigma \cup \{[\!],]\!]\}, P_k, S)$ which generates $s(D(G_k))$ as follows, where we will use two specific symbols $[\!]$ and $]\!]$ to mark the production rules that connect G_k to $B^{(r)}$.

We consider each splicing rule $r = u_1(\#(u_2))\$u_3(\#(u_4))$ for which both $u_1(u_2)$ and $u_3(u_4)$ occur as subtrees of elements of $s(D(G_k))$. For each such rule we consider all derivation trees of G_k which contain $u_1(u_2)$ or $u_3(u_4)$ as subtrees. If $\alpha_0(\phi_1(\alpha_1, \ldots, \phi_2(\alpha'_1, \ldots, \alpha'_{m_2}), \ldots, \alpha_{m_1}))$ is a derivation tree of G_k such that

$$s(\phi_1(\lambda_1, \ldots, \phi_2(\lambda'_1, \ldots, \lambda'_{m_2}), \ldots, \lambda_{m_1})) = u_1(\lambda_1, \ldots, u_2(\lambda'_1, \ldots, \lambda'_{m_2}), \ldots, \lambda_{m_1})$$

(where λ_i and λ'_j are null trees) and the root of ϕ_1 is a nonterminal X and the root of α_i is a nonterminal Y_i for $1 \leqslant i \leqslant m_1$, then we add a new production rule $X \to [\![S^{(r)}$ and new production rules $Z_i^{(r)} \to]\!]Y_i$ where $S^{(r)}, Z_i^{(r)} \in N^{(r)}$ $(1 \leqslant i \leqslant m_1)$. Similarly, if there is a derivation tree $\beta_0(\phi_3(\beta_1, \ldots, \phi_4(\beta'_1, \ldots, \beta'_{n_2}), \ldots, \beta_{n_1}))$ of G_k such that $s(\phi_3(\phi_4)) = u_3(u_4)$ and the root of β'_i is a nonterminal Y'_i for $1 \leqslant i \leqslant n_2$, then we add new production rules $\hat{Z}_i^{(r)} \to]\!]Y'_i$ $(1 \leqslant i \leqslant n_2)$. A production rule of the form $X \to [\![S^{(r)}$ is called an *initial marker-production of level* $k+1$ if it is not included in G_k, and a production rule of the form $Z \to]\!]Y$ is called an *terminal marker-production of level* $k+1$ if it is not included in G_k. If the set of production rules $P^{(r)}$ is not included in G_k, then we say that $B^{(r)}$ has level $k+1$.

We obtain G_{k+1} from G_k by adding all marker-productions and CFGs $B^{(r)}$ of level $k+1$ except production rules $Z_i^{(r)} \to \lambda_i$ and $\hat{Z}_j^{(r)} \to \lambda'_j$ (that is, $P^{(r)} - \{Z_i^{(r)} \to \lambda_i, \hat{Z}_j^{(r)} \to \lambda'_j \mid 1 \leqslant i \leqslant m_1, 1 \leqslant j \leqslant n_2\}$).

Since the number of possible marker-productions is finite, eventually there is a level n for which $G_{n+1} = G_n$. Now we will show that the tree language $s(D(G_n))$ is closed under splicing and that $s(D(G_n))$ is included in $\gamma^*(L_0)$. We extend the mapping s to $s(X[\![t) = s(t)$ and $s(Z]\!]t) = s(t)$ for dealing with marker-productions where X and Z are nonterminals.

To see closure under splicing, suppose that two skeletons

$$x_0(u_1(x_1, \ldots, u_2(x'_1, \ldots, x'_{m_2}), \ldots, x_{m_1})) \quad \text{and} \quad y_0(u_3(y_1, \ldots, u_4(y'_1, \ldots, y'_{n_2}), \ldots, y_{n_1}))$$

are in $s(D(G_n))$ and that a splicing rule r

$$r = u_1(\lambda_1,\ldots,\#(u_2(\lambda'_1,\ldots,\lambda'_{m_2})),\ldots,\lambda_{m_1}) \$ u_3(\lambda_1,\ldots,\#(u_4(\lambda'_1,\ldots,\lambda'_{n_2})),\ldots,\lambda_{n_1})$$

is in R. Note that $b_1^{(r)}(b_4^{(r)}) \in D(B^{(r)})$ and $s(b_1^{(r)}(b_4^{(r)})) = u_1(u_4)$. Let

$$\alpha_0(\phi_1(\alpha_1,\ldots,\phi_2(\alpha'_1,\ldots,\alpha'_{m_2}),\ldots,\alpha_{m_1})) \quad \text{and} \quad \beta_0(\phi_3(\beta_1,\ldots,\phi_4(\beta'_1,\ldots,\beta'_{n_2}),\ldots,\beta_{n_1}))$$

be derivation trees of G_n such that $s(\alpha_0)=x_0$, $s(\phi_1(\phi_2))=u_1(u_2)$, $s(\alpha_i)=x_i$ $(1\leqslant i\leqslant m_1)$ and $s(\alpha'_j)=x'_j$ $(1\leqslant j\leqslant m_2)$, and similarly for β_0, ϕ_3, ϕ_4, β_i, β'_j. If the root of ϕ_1 is a nonterminal X then there is an initial marker-production $X \to [\![S^{(r)}$, and if the root of α_i is a nonterminal Y_i then there are terminal marker-productions $Z_i^{(r)} \to]\!]Y_i$ $(1\leqslant i\leqslant m_1)$, and if the root of β'_i is a nonterminal Y'_i then there are terminal marker-productions $\hat{Z}_i^{(r)} \to]\!]Y'_i$ $(1\leqslant i\leqslant n_2)$. Hence

$$\alpha_0(X[\![b_1^{(r)}(Z_1^{(r)}]\!]\alpha_1,\ldots,b_4^{(r)}(\hat{Z}_1^{(r)}]\!]\beta'_1,\ldots,\hat{Z}_{n_2}^{(r)}]\!]\beta'_{n_2}),\ldots,Z_{m_1}^{(r)}]\!]\alpha_{m_1}))$$

is a derivation tree of G_n and so

$$x_0(u_1(x_1,\ldots,u_4(y'_1,\ldots,y'_{n_2}),\ldots,x_{m_1})) = s(\alpha_0(X[\![b_1^{(r)}(Z_1^{(r)}]\!]\alpha_1,\ldots,b_4^{(r)}(\hat{Z}_1^{(r)}]\!]\beta'_1,\ldots,\hat{Z}_{n_2}^{(r)}]\!]\beta'_{n_2}),\ldots,Z_{m_1}^{(r)}]\!]\alpha_{m_1})))$$

is in $s(D(G_n))$.

To show that $s(D(G_n))$ is included in $\gamma^*(L_0)$, we will show how to obtain $s(t) \in s(D(G_n))$ by a finite number of splicings starting with elements of L_0.

We use the following notion defined by Pixton [7]. The *complexity* of a derivation tree t of G_n is the n-tuple of natural numbers $c = \langle c_1,\ldots,c_n\rangle \in \mathbb{N}^n$, where c_j is the number of marker-productions used in t of level j. We order the set of these tuples lexicographically. The proof is constructed based on induction on the partial order of this complexity.

First, a derivation tree t of complexity $0 = \langle 0,\ldots,0\rangle$ is generated by G_0, so $s(t)$ is in L_0.

Suppose that $c \in \mathbb{N}^n$ with $c > 0$ and suppose that for all derivation trees t' of G_n of complexity less than c, $s(t')$ is in $\gamma^*(L_0)$. Suppose that a derivation tree t of G_n has complexity c. Then t starts with the start symbol S, and t contains at least one marker-production and furthermore it contains at least one initial marker-production. We select an initial marker-production $X \to [\![S'$ such that

$$\xi(\ X[\![S'(\ w\ )\ ) = t$$

and w contains no initial marker-production. Since w contains some terminal marker-productions, we select terminal marker-productions $Z_i \to]\!]Y_i$ such that

$$\theta_1(\ Z_1]\!]Y_1(\eta_1),\ldots,\theta_2(\ Z'_1]\!]Y'_1(\eta'_1),\ldots,Z'_{q_2}]\!]Y'_{q_2}(\eta'_{q_2})\ ),\ldots,Z_{q_1}]\!]Y_{q_1}(\eta_{q_1})\ ) = w$$

and $\theta_1(\theta_2)$ contains no terminal marker-production. Note that

$$t = \xi(X[\![S'(\theta_1(Z_1]\!]Y_1(\eta_1),\ldots,\theta_2(Z'_1]\!]Y'_1(\eta'_1),\ldots,Z'_{q_2}]\!]Y'_{q_2}(\eta'_{q_2})),\ldots,Z_{q_1}]\!]Y_{q_1}(\eta_{q_1})))).$$

Now we can see that there is a splicing rule $r = u_1(\#(u_2))\$u_3(\#(u_4))$ such that

$$b_1^{(r)}(\lambda,\ldots,b_4^{(r)}(\lambda,\ldots,\lambda),\ldots,\lambda) = S'(\theta_1(Z_1(\lambda),\ldots,\theta_2(Z_1'(\lambda),\ldots,Z_{q_2}'(\lambda)),\ldots,Z_{q_1}(\lambda)))$$

and $s(b_1^{(r)}(b_4^{(r)})) = u_1(u_4)$.

Suppose that $X \to [\![S'$ has level k and $S' = S^{(r)}$, and each $Z_i \to]Y_i$ has level at most k' and $Z_i = Z_i^{(r)}$ $(1 \leqslant i \leqslant q_1)$, and each $Z_j' \to]Y_j'$ has level at most k' and $Z_j' = \hat{Z}_j^{(r)}$ $(1 \leqslant j \leqslant q_2)$. Then there is a derivation tree

$$\alpha_0(\phi_1(\alpha_1,\ldots,\phi_2(\alpha_1',\ldots,\alpha_l'),\ldots,\alpha_{q_1}))$$

in G_{k-1} such that the root of ϕ_1 is a nonterminal X, the root of α_i is a nonterminal Y_i $(1 \leqslant i \leqslant q_1)$, and $s(\phi_1(\phi_2)) = u_1(u_2)$, and there is a derivation tree

$$\beta_0(\phi_3(\beta_1,\ldots,\phi_4(\beta_1',\ldots,\beta_{q_2}'),\ldots,\beta_m))$$

in $G_{k'-1}$ such that the root of β_j' is a nonterminal Y_j' $(1 \leqslant j \leqslant q_2)$ and $s(\phi_3(\phi_4)) = u_3(u_4)$. It follows that

$$\xi(\phi_1(Y_1(\eta_1),\ldots,\phi_2(\alpha_1',\ldots,\alpha_l'),\ldots,Y_{q_1}(\eta_{q_1}))) = t_1$$

and

$$\beta_0(\phi_3(\beta_1,\ldots,\phi_4(Y_1'(\eta_1'),\ldots,Y_{q_2}'(\eta_{q_2}')),\ldots,\beta_m)) = t_2$$

are derivation trees in G_n, and that

$$s(t_1) = s(\xi)(u_1(s(Y_1(\eta_1)),\ldots,u_2(s(\alpha_1'),\ldots,s(\alpha_l')),\ldots,s(Y_{q_1}(\eta_{q_1}))))$$

and

$$s(t_2) = s(\beta_0)(u_3(s(\beta_1),\ldots,u_4(s(Y_1'(\eta_1')),\ldots,s(Y_{q_2}'(\eta_{q_2}'))),\ldots,s(\beta_m)))$$

splice together using the rule r to give

$$\begin{aligned} &s(\xi)(u_1(s(Y_1(\eta_1)),\ldots,u_4(s(Y_1'(\eta_1')),\ldots,s(Y_{q_2}'(\eta_{q_2}'))),\ldots,s(Y_{q_1}(\eta_{q_1})))) \\ &\quad = s(\xi(b_1^{(r)}(Y_1(\eta_1),\ldots,b_4^{(r)}(Y_1'(\eta_1'),\ldots,Y_{q_2}'(\eta_{q_2}')),\ldots,Y_{q_1}(\eta_{q_1}))) \\ &\quad = s(t). \end{aligned}$$

We can see that t_1 and t_2 have lower complexities than t. By the induction hypothesis, both $s(t_1)$ and $s(t_2)$ are in $\gamma^*(L_0)$, and hence $s(t)$ is also in $\gamma^*(L_0)$. This concludes the proof. □

Corollary 4. *A skeleton language in the class $HTS(FIN_T, FIN_R)$ is generated by a context-free grammar.*

5. Conclusions

It is known that splicing systems, with finite set of rules, and finite set of starting strings, can generate only regular languages. As soon as we can use multiplicity constraints on the same class of splicing systems, we are able to generate any recursively enumerable languages. In this paper we looked for variations, other than multiplicity, to the basic definition of splicing system, with the goal of going beyond the generation of regular languages.

We defined a class of splicing systems able to generate context-free languages, or structures closely related to them, like derivation trees or skeletons. An interesting feature is that our model keeps good molecular feasibility, since it generates folded strings similar to the representation in terms of secondary structures of RNA molecules.

We could even consider directly the implementation of our splicing systems by DNA, since it has been proved that it can be assembled in unusual spatial structures, using strands partially annealed as we also propose. See for instance the solid-support based methodology to build complex DNA structures defined in [11], but also the suggestive comments on the experimental pitfalls of this constructions reported in [9].

Recently, Winfree et al. discussed in [10] a way of generating context-free languages similar to ours: it is based on the iterated joining of short double stranded sequences, reproducing in the finally resulting molecule all the derivation of a string. With respect to that paper, the main advantage of our proposal is that while they encode in the molecule all the terminals *and* the nonterminals involved in the derivation, here we can work with cleaner structures, like those representing the skeletons. We can simulate a correct derivation of trees by splicing even without keeping record of nonterminals, since the restriction enzymes recognize specific substructures in the molecules, and not directly the nonterminals, enough to allow them to splice two trees and so to continue the derivation process.

Acknowledgement

This work is supported in part by "Research for the Future" Program No. JSPS-RFTF 96I00101 from the Japan Society for the Promotion of Science. We would like to thank anonymous referees for their valuable comments which improved the paper very much.

References

[1] L. Adleman, Molecular computation of solutions to combinatorial problems, Science 266 (1994) 1021–1024.

[2] R. Freund, L. Kari, Gh. Paun, DNA computing based on splicing: the existence of universal computers, Technical report, Fachgruppe Informatik, Tech. Univ., Wien.

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

[4] J.R. Koza, Genetic Programming, MIT Press, Cambridge, MA, 1992.
[5] G. Paun, On the splicing operation, Discrete Appl. Math. 70 (1996) 57–79.
[6] Gh. Paun, G. Rozenberg, A. Salomaa, Computing by splicing, Theoret. Comput. Sci. 168 (2) (1996) 321–336.
[7] D. Pixton, Regularity of splicing languages, Discrete Appl. Math. 69 (1996) 101–124.
[8] Y. Sakakibara, M. Brown, R. Hughey, I.S. Mian, K. Sjolander, R.C. Underwood, D. Haussler, Stochastic context-free grammars for tRNA modeling, Nucleic Acids Res. 22 (1994) 5112–5120.
[9] N.C. Seeman, H. Wang, B. Liu, J. Qi, X. Li, X. Yang, F. Liu, W. Sun, Z. Shen, R. Sha, C. Mao, Y. Wang, S. Zhang, T-J. Fu, S. Du, J.E. Mueller, Y. Zhang, J. Chen, The perils of polynucleotides: the experimental gap between the design and assembly of unusual DNA structures, in: Proc. 2nd Meeting on DNA Based Computers, Princeton, 1996; AMS–DIMACS Series in Discrete. Math. and Theoret. Comput. Sci., to appear.
[10] E. Winfree, X. Yang, N.C. Seeman, Universal computation via self-assembly of DNA: some theory and experiments, in: Proc. 2nd Meeting on DNA Based Computers, Princeton, 1996; in AMS–DIMACS Series in Discr. Math. and Theoret. Comput. Sci., to appear.
[11] Y. Zhang, N.C. Seeman, A solid-support methodology for the construction of geometrical objects from DNA, J. Am. Chem. Soc. 114 (1992) 2656–2663.