# Upper Bounds for Restricted Splicing*

Hendrik Jan Hoogeboom and Nikè van Vugt

Universiteit Leiden, Institute of Advanced Computer Science
P.O. Box 9512, 2300 RA Leiden, The Netherlands

**Abstract.** We determine or improve upper bounds for non-iterated splicing in length-increasing, length-decreasing, same-length and self splicing mode.

## 1 Introduction

The cutting and recombination of DNA with the help of restriction enzymes has been abstracted as the splicing operation for formal languages, see for instance the introduction by Head, Păun, and Pixton [HPP97], or the relevant chapters in the book of Păun, Rozenberg, and Salomaa on computational models inspired by DNA computing [PRS98].

The splicing operation takes two strings, and cuts them in a position specified by a splicing rule. Then these strings are recombined after exchanging their postfixes (the parts of the strings following the cut). This operation can then be studied within the framework of formal language theory, in order to estimate its computational power. One may study its effect as a closure operation on language families, or one may study its power when applied iteratively as if it were a single step of a computing device. Most famous in this latter area is the result that the family of regular languages is closed under splicing (using a finite set of rules) [CH91]. In fact, for the Chomsky hierarchy the power of splicing has been extensively investigated, and optimal upper bounds within the hierarchy have been established (cf. [HPP97], or [PRS98]).

Here we concentrate on the non-iterative, single, application of the splicing operation applied to families of the Chomsky hierachy. What is open here is the power of some modes of restricted splicing, i.e., splicing where there are additional constraints on the two strings involved, as inspired by [PRS96,KPS96]. For instance, in same-length splicing both strings are required to be of the same length, and in self splicing both strings are assumed to be identical.

In particular, it is left open whether non-iterated splicing in one of the modes length-decreasing, same-length, and self splicing, stays within the context-sensitive languages when applied to regular languages, using context-free sets of rules. We will show that this is indeed the case for same-length and length-decreasing mode (Corollaries 8(1) and 12(1)), whereas self splicing generates every recursively enumarable language (up to a single marker, Corollary 16).

* July 2001, LIACS Technical Report 01-05, `www.liacs.nl`, submitted.

Moreover, we show that applying either of these splicing modes to context-free languages with finite or regular sets of rules, results in a context-sensitive language. This was open for same-length splicing (Corollary 8(2)) and self splicing (Corollary 14).

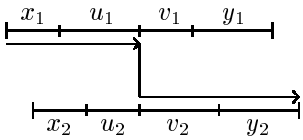
After defining the operation of splicing in Section 2, we explain our basic tools in Section 3. In the next three sections we discuss same-length splicing, length-increasing (and length-decreasing) splicing, and self splicing, respectively.

## 2 Splicing

A *splicing rule* over an alphabet $V$ is an element of $(V^*)^4$. For such a rule $r = (u_1, v_1, u_2, v_2)$ and strings $x, y, z \in V^*$ we write

$$(x, y) \vdash_r z \text{ iff } x = x_1 u_1 v_1 y_1,\ y = x_2 u_2 v_2 y_2 \text{ and}$$
$$z = x_1 u_1 v_2 y_2, \text{ for some } x_1, y_1, x_2, y_2 \in V^*.$$

We say that the string $z$ is obtained by *splicing* the strings $x$ and $y$ *using* the rule $r$.




A *splicing system* (or *H system*) is a triple $h = (V, L, R)$ where $V$ is an alphabet, $L \subseteq V^*$ is the *initial language* and $R \subseteq (V^*)^4$ is a set of splicing rules, the *splicing relation*.

To estimate the complexity of sets of rules using the familiar Chomsky hierarchy as a yard stick, splicing rules are commonly represented as strings rather than 4-tuples: a splicing rule $r = (u_1, v_1, u_2, v_2)$ is given as the string $\mathsf{Z}(r) = u_1 \# v_1 \$ u_2 \# v_2$ (# and \$ are special symbols not in $V$), i.e., $\mathsf{Z}$ is a mapping from $(V^*)^4$ to $V^*\#V^*\$V^*\#V^*$, that gives a *string representation* of each splicing rule. We extend $\mathsf{Z}$ in the natural way to a mapping from sets of splicing rules to languages. In agreement with this policy, we usually write, for instance, "regular set of splicing rules" when we mean a set of splicing rules of which the $\mathsf{Z}$-representation is a regular language. It was argued in [HvV98] that this representation is quite robust: most of the other, related, representations do not change the position in the Chomsky hierarchy of the families resulting from (uniterated) splicing $\mathcal{F}_1$-languages using $\mathcal{F}_2$-rules.

In this paper we consider the setting where the general splicing operation $(x, y) \vdash_r z$ may only be applied in a certain context. We recall the definitions of certain types of restricted splicing from [PRS96,KPS96]. We splice in (*length-*) *increasing* mode (*in* for short) if the length of the resulting string is strictly greater than the lengths of the two input strings, in (*length-*) *decreasing* mode (*de*) if the length of the resulting string is strictly smaller than the lengths of

the two input strings, in *same-length* mode ($sl$) if the two input strings have the same length, and in *self splicing* mode ($sf$) if the two input strings are equal. We add *free* splicing ($f$) as a synonym for unrestricted splicing. Formally, for a splicing rule $r$ we use the following relations.

| | | |
|---|---|---|
| *free* | $(x,y) \vdash_r^f z$ | iff $(x,y) \vdash_r z$ unrestricted |
| *increasing* | $(x,y) \vdash_r^{in} z$ | iff $(x,y) \vdash_r z$ and $\lvert z\rvert > \max\{\lvert x\rvert, \lvert y\rvert\}$ |
| *decreasing* | $(x,y) \vdash_r^{de} z$ | iff $(x,y) \vdash_r z$ and $\lvert z\rvert < \min\{\lvert x\rvert, \lvert y\rvert\}$ |
| *same-length* | $(x,y) \vdash_r^{sl} z$ | iff $(x,y) \vdash_r z$ and $\lvert x\rvert = \lvert y\rvert$ |
| *self* | $(x,y) \vdash_r^{sf} z$ | iff $(x,y) \vdash_r z$ and $x = y$. |

Note that the requirement for length-increasing splicing can be reformulated in terms of the two input strings $x$ and $y$, without explicitly mentioning the result $z$ of the splicing. If we splice $x = x_1u_1v_1y_1$ and $y = x_2u_2v_2y_2$ using the rule $r = (u_1, v_1, u_2, v_2)$, it is in increasing mode iff $|x_1u_1| > |x_2u_2|$ and $|v_1y_1| < |v_2y_2|$. There is a similar formulation for length-decreasing splicing.

Let $h = (V, L, R)$ be a splicing system. With the splicing modes given above we define the (non-iterated splicing) languages

$$\sigma_\mu(h) = \{\ z \in V^* \mid (x,y) \vdash_r^\mu z \text{ for some } x, y \in L \text{ and } r \in R\ \}$$

for $\mu \in \{f, in, de, sl, sf\}$. Similarly we define the families

$$S_\mu(\mathcal{F}_1, \mathcal{F}_2) = \{\ \sigma_\mu(h) \mid h = (V, L, R) \text{ with } L \in \mathcal{F}_1 \text{ and } \mathsf{Z}(R) \in \mathcal{F}_2\ \}.$$

A splicing system with $L \in \mathcal{F}_1$ and $\mathsf{Z}(R) \in \mathcal{F}_2$ is said to be of $(\mathcal{F}_1, \mathcal{F}_2)$ type.

**Example 1.** Let $h = (\{a, b, c\}, L, R)$ be the splicing system defined by

$$L = c\, b^*a^*b^*c$$
$$R = \{\ (cb^ma^n, b^nc, c, b^mc) \mid m, n \geq 0\}$$

It is of (REG,LIN) type as the initial language $L$ is regular, and the set of rules is linear: $\mathsf{Z}(R) = \{cb^ma^n\#b^nc\$c\#b^mc \mid m, n \geq 0\}$.

The only splicings possible are of the form $(cb^ma^n \mid b^nc, c \mid b^mc) \vdash cb^ma^nb^mc$. If the splicing has to be done in length-increasing mode, then we must have $m + n + 1 > 1$ and $m + 1 > n + 1$, hence

$$\sigma_{in}(h) = \{cb^ma^nb^mc \mid m, n \geq 0 \text{ and } m > n\}$$

which is not a context-free language. □

We only consider $\mathcal{F}_1 = \mathsf{REG}, \mathsf{LIN}, \mathsf{CF}$ and $\mathcal{F}_2 = \mathsf{FIN}, \mathsf{REG}, \mathsf{LIN}, \mathsf{CF}$, and in particular we are interested in upper bounds for the families $S_\mu(\mathcal{F}_1, \mathcal{F}_2)$ for the modes $\mu$ that we consider. Known results are from the papers [PRS98], which deals with finite sets of rules ($\mathcal{F}_2 = \mathsf{FIN}$) only, and [KPS96], which deals with sets of rules of arbitrary Chomsky complexity (contradicting its title). We repeat in Table 1 the parts of Tables 1, 2 and 3 from [KPS96] that summarize the lowest

| | FIN | REG | LIN | CF | FIN | REG | LIN | CF | FIN | REG | LIN | CF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $f$ | REG | REG | LIN | CF | CF | CF | RE | RE | CF | CF | RE | RE |
| $in$ | REG | REG | | CF+ | CS | CS | CS | CS | CS | CS | CS | CS |
| $de$ | REG | REG | | CF+ | CS | CS | RE | RE | CS | CS | RE | RE |
| $sl$ | LIN | LIN | CF+ | | CF+ | | RE | RE | CF+ | | RE | RE |
| $sf$ | CS | CS | | | CF+ | | RE | RE | CF+ | | RE | RE |
| | $\mathcal{F}_1 = \mathsf{REG}$ | | | | $\mathcal{F}_1 = \mathsf{LIN}$ | | | | $\mathcal{F}_1 = \mathsf{CF}$ | | | |

**Table 1.** Upper bounds of $S_\mu(\mathcal{F}_1, \mathcal{F}_2)$ within the Chomsky hierarchy [KPS96].

upper bounds within the Chomsky hierarchy for the families $S_\mu(\mathcal{F}_1, \mathcal{F}_2)$. The families of initial languages $\mathcal{F}_1$ are listed in the bottom row, the families of splicing rules $\mathcal{F}_2$ are in the top row, repeated for each of the three possible initial families $\mathcal{F}_1$.

For the entries marked with CF+ it is only known that the family contains a non-context-free language; it is not yet determined whether the smallest upper bound within the Chomsky hierarchy is CS or RE.

Note that although, for instance, the table contains the same bounds for the families $S_{sl}(\mathsf{REG}, \mathsf{FIN})$ and $S_{sl}(\mathsf{REG}, \mathsf{REG})$, this does not necessarily mean that they are equal: they only have the same upper bound in the Chomsky hierarchy. The same remark holds for the equality of the tables for $\mathcal{F}_1 = \mathsf{LIN}$ and $\mathcal{F}_1 = \mathsf{CF}$.

## 3 Basic tools

We present our basic tools. First we define a language that captures both the initial language and the rules of a splicing system. Second, we recall the notion of valence grammar, a grammatical device modestly extending the context-free grammars.

### 3.1 Representing the system by a language

The open problems indicated in Table 1 involve either a context-free (or even linear) initial language, and regular (or even finite) splicing rules, or vice versa. For unrestricted (free) non-iterated splicing the upper bounds for these two cases are determined in Lemma 3.3 and Lemma 3.6 of [HPP97]. We use the ideas from the proofs of these two lemma's to define, for each splicing system $h = (V, L, R)$, the language $C(L, R)$ which combines the initial language with the rules.

$$C(L,R) = \{\ x_1u_1\#v_1y_1\$x_2u_2\#v_2y_2 \ \mid x_1u_1v_1y_1\ ,\ x_2u_2v_2y_2 \in L \text{ and } (u_1, v_1, u_2, v_2) \in R\}$$

This language turns out to be very helpful in determining upper bounds for (restricted) splicing families. Note that $\sigma_f(h) = \text{join}(C(L, R))$, where join is the finite state transduction that erases the two #'s, and everything in between, from a string in $C(L, R)$.

To construct $C(L,R)$ from $L$ and $R$, we proceed as follows. It is straightforward to design a (non-deterministic) finite state transduction such that the language $\mathsf{Z}(R)$, representing the rules, is transformed into the language $R' = \{x_1u_1\#v_1y_1\$x_2u_2\#v_2y_2 \mid u_1\#v_1\$u_2\#v_2 \in \mathsf{Z}(R) \text{ and } x_1,y_1,x_2,y_2 \in V^*\}$. Also using a finite state transduction, the language $L' = \{x\#y\$w\#z \mid xy\ ,\ wz \in L\}$ can be constructed from $L \cdot \$ \cdot L$, the (marked) concatenation of the initial language $L$ with itself. Clearly, $C(L,R) = L' \cap R'$.

Since both $\mathsf{REG}$ and $\mathsf{CF}$ are closed under finite state transductions and under concatenation, we either have $L' \in \mathsf{REG}$ and $R' \in \mathsf{CF}$ for splicing of ($\mathsf{REG}$,$\mathsf{CF}$) type, or $L' \in \mathsf{CF}$ and $R' \in \mathsf{REG}$, for splicing of ($\mathsf{CF}$,$\mathsf{REG}$) type. Clearly in both cases $C(L,R) = L' \cap R'$ is a context-free language, and so is $\sigma_f(h) = \mathrm{join}(C(L,R))$, proving the (known) upper bounds for ($\mathsf{REG}$,$\mathsf{CF}$) type and ($\mathsf{CF}$,$\mathsf{REG}$) type splicing.

**Lemma 2.** *Let $h = (V,L,R)$ be a splicing system of $(\mathsf{REG},\mathsf{CF})$ type or $(\mathsf{CF},\mathsf{REG})$ type. Then the language $C(L,R)$ is context-free.*

In the sequel we adapt this strategy to restricted splicing. In that case we have to put further restrictions on the pair of strings that is spliced. This leads us to consider particular subsets of the language $C(L,R)$.

In the case of same-length splicing for instance, we have to restrict ourselves to strings in $C(L,R)$ for which additionally $|x_1u_1v_1y_1| = |x_2u_2v_2y_2|$. The resulting language which we call $C_{sl}(L,R)$ again represents the system in the sense that $\sigma_{sl}(h) = \mathrm{join}(C_{sl}(L,R))$. In the case of $(\mathsf{REG},\mathsf{CF})$ type splicing, $C_{sl}(L,R)$ is in general no longer context-free, but context-sensitive. This means that the upper bound we obtain in this way for $S_{sl}(\mathsf{REG},\mathsf{CF})$, i.e., by applying a finite state transduction to $C_{sl}(L,R)$, is $\mathsf{RE}$ rather than $\mathsf{CS}$, as $\mathsf{CS}$ is not closed under finite state transductions (in particular it is not closed under erasing mappings).

However, it turns out that $S_{sl}(\mathsf{REG},\mathsf{CF}) \subseteq \mathsf{RE}$ is not the optimal upper bound that is valid within the Chomsky hierarchy. Hence the applicability of our method fails because of the poor closure properties of $\mathsf{CS}$, and we have been looking for a natural family strictly in between $\mathsf{CF}$ and $\mathsf{CS}$, closed under finite state transductions, and which contains the languages $C_\mu(L,R)$ for the splicing types $\mu$ we will consider. Such a family exists, and we discuss its characteristics in the next subsection.

### 3.2 Valence grammars

Let $k \geq 1$. We use $\mathbb{Z}^k$ to denote the set of $k$-dimensional vectors over the integers, and the vector with all components zero is written as $\mathbf{0}$.

A *context-free valence grammar* over $\mathbb{Z}^k$ is a context-free grammar in which every production is assigned a vector from $\mathbb{Z}^k$, the valence of the production. A string belongs to the language of the grammar if it is derived in the usual, context-free, fashion, while additionally the valences of the productions used add up to zero.

Formally, such a grammar is a construct $G = (\Sigma,\Delta,R,S)$, where $\Sigma$ is the alphabet, $\Delta \subseteq \Sigma$ is the terminal alphabet, $S \in \Sigma$ is the axiom, and $R$ is a finite

set of *rules*, each of which is of the form $[\pi, \boldsymbol{r}]$, where $\pi \in (\Sigma - \Delta) \times \Sigma^*$ is a context-free production, and $\boldsymbol{r} \in \mathbb{Z}^k$ is its associated valence.

For $x, y \in \Sigma^*$ and $\boldsymbol{v} \in \mathbb{Z}^k$ we write $(x, \boldsymbol{v}) \Rightarrow_G (y, \boldsymbol{v} + \boldsymbol{r})$ if there exist a rule $[A \to z, \boldsymbol{r}]$ and $x_1, x_2 \in \Sigma^*$, such that $x = x_1 A x_2$ and $y = x_1 z x_2$. The *language generated by* $G$ equals $L(G) = \{ w \in \Delta^* \mid (S, \mathbf{0}) \Rightarrow^* (w, \mathbf{0}) \}$.

The resulting family of valence languages over $\mathbb{Z}^k$ is denoted by $\mathsf{CF}(\mathbb{Z}^k)$. Valence grammars were introduced in [Pău80]. A good starting point for learning of their properties and for pointers to recent literature is the paper [FS00], presented as extended abstract in [FS97].

**Example 3.** (1) Consider the valence grammar $G_1$ over $\mathbb{Z}^2$, which has rules $[S \to aS, (+1, +1)]$, $[S \to bS, (-1, 0)]$, $[S \to cS, (0, -1)]$, and $[S \to \lambda, (0, 0)]$, where $S$ is the axiom, and $a, b, c$ are terminal symbols. Then $L(G_1) = \{ w \in \{a, b, c\}^* \mid \#_a(w) = \#_b(w) = \#_c(w) \}$, as the first component of the valence forces $\#_a(w) = \#_b(w)$, while $\#_a(w) = \#_c(w)$ because of the second component. This is in fact a right-linear valence grammar.

(2) The same language can be obtained by a valence grammar over $\mathbb{Z}^1$ choosing rules $[S \to SS, (0)]$, $[S \to aSb, (+1)]$, $[S \to bSa, (+1)]$, $[S \to cS, (-1)]$, and $[S \to \lambda, (0)]$.

Note that this is essentially a context-free grammar for the language $\{ w \in \{a, b, c\}^* \mid \#_a(w) = \#_b(w) \}$, augmented with an additional counter to compare the number of $c$'s to the numbers of $a$'s and $b$'s. □

The right-linear valence grammars are a formalism equivalent to the blind counter automata of Greibach [Gre78]; these are finite state automata equipped with additional counters, each of which can be incremented and decremented independently. This storage is "blind" as the counters cannot be tested for zero during the computation, except implicitly at the end, as one only considers computations that lead from the initial state with empty counters to an accepting state with empty counters.

The context-free valence languages form a hierarchy within $\mathsf{CS}$. Each $\mathsf{CF}(\mathbb{Z}^k)$ has very convenient closure properties; it is in fact a full semi-AFL.

**Proposition 4.** *Let $k \geq 1$. $\mathsf{CF}(\mathbb{Z}^k)$ is closed under union, homomorphisms, inverse homomorphisms, and intersection with regular languages. Consequently $\mathsf{CF}(\mathbb{Z}^k)$ is closed under finite state transductions.*

The closure under intersection with regular languages can be generalized as follows: the intersection of a context-free valence language over $\mathbb{Z}^k$ with a right-linear valence language over $\mathbb{Z}^\ell$ is a context-free valence language over $\mathbb{Z}^{k+\ell}$. We will use this fact in the sequel, in particular for $k = 0$, i.e., the intersection of a context-free language and a right-linear valence language over $\mathbb{Z}^\ell$ belongs to $\mathsf{CF}(\mathbb{Z}^\ell)$.

We end by giving two more examples of right-linear valence languages which are essential for our considerations.

**Example 5.** Let $\Sigma$ be an alphabet, and let $\#, \$$ be two symbols not in $\Sigma$.

(1) Consider $D_{sl} = \{\ x_1\#y_1\$x_2\#y_2 \mid x_1, y_1, x_2, y_2 \in \Sigma^*, |x_1y_1| = |x_2y_2|\ \}$. It is generated by a right-linear valence grammar over $\mathbb{Z}^1$, with axiom $S_0$, and the following rules. Here $a$ ranges over $\Sigma$.

$[S_0 \to aS_0, (+1)\,]$, $[S_0 \to \#S_1, (0)\,]$,
$[S_1 \to aS_1, (+1)\,]$, $[S_1 \to \$S_2, (0)\,]$,
$[S_2 \to aS_2, (-1)\,]$, $[S_2 \to \#S_3, (0)\,]$, and
$[S_3 \to aS_3, (-1)\,]$, $[S_3 \to \lambda, (0)\,]$

(2) $D_{in} = \{\ x_1\#y_1\$x_2\#y_2 \mid x_1, y_1, x_2, y_2 \in \Sigma^*, |x_1| > |x_2|$ and $|y_2| > |y_1|\ \}$ is generated by the right-linear valence grammar over $\mathbb{Z}^2$, with axiom $S_0$, and the following rules; again, $a$ ranges over $\Sigma$.

$[S_0 \to aS_0, (+1, 0)\,]$, $[S_0 \to aS_0, (0, 0)\,]$, $[S_0 \to \#S_1, (0, 0)\,]$,
$[S_1 \to aS_1, (0, -1)\,]$, $[S_1 \to \$S_2, (0, 0)\,]$,
$[S_2 \to aS_2, (-1, 0)\,]$, $[S_2 \to \#S_3, (0, 0)\,]$, and
$[S_3 \to aS_3, (0, +1)\,]$, $[S_3 \to aS_3, (0, 0)\,]$, $[S_3 \to \lambda, (-1, -1)\,]$.

Observe that we have inequality $|x_1| \geq |x_2|$ rather than equality because symbols in the first segment do not have to be counted on the (first) counter as there is an alternative rule. The strictness of the inequality is forced by decreasing both counters in the final rule. □

## 4 Same-length splicing

We restrict splicing to cases where both inputs have the same length, i.e., mode $\mu = sl$. Precise upper bounds within the Chomsky hierarchy are missing for $S_{sl}(\mathcal{F}_1, \mathcal{F}_2)$ when (1) $\mathcal{F}_1 = \mathsf{REG}$ and $\mathcal{F}_2 = \mathsf{LIN}, \mathsf{CF}$ (two cases), and when (2) $\mathcal{F}_1 = \mathsf{LIN}, \mathsf{CF}$ and $\mathcal{F}_2 = \mathsf{FIN}, \mathsf{REG}$ (four cases).

All these families contain a non-context-free language. For $S_{sl}(\mathsf{REG}, \mathsf{LIN})$ this was shown in [KPS96, Lemma 8]; for $S_{sl}(\mathsf{LIN}, \mathsf{FIN})$ this follows from the fact that one may closely simulate the operation of doubling using splicing with finite rules, cf. Lemma 3 in [PRS98], obtaining from $L$ the language double$(L) = \{xx \mid x \in L\}$.

We give an explicit example for the latter family $S_{sl}(\mathsf{LIN}, \mathsf{FIN})$.

**Example 6.** Let $h = (\{a, b, d\}, L, R)$ be the splicing system of (LIN,FIN) type defined by

$$L = \{a^n b^n d \mid n \geq 1\} \cup \{d\, a^n b^n \mid n \geq 1\}$$
$$R = \{\ (b, d, d, a)\ \}$$

The form of the rule causes the first argument of each splicing to be of the form $a^k b^k d$, and the second argument of the form $d\, a^j b^j$, for some $k, j \geq 1$. Moreover, if we consider same-length splicing, we should have $k = j$. Then $(a^k b^k \mid d,\ d \mid a^k b^k) \vdash^{sl} a^k b^k a^k b^k$, using the only splicing rule in $R$. Consequently

$$\sigma_{sl}(h) = \{a^n b^n a^n b^n \mid n \geq 1\}$$

which is not context-free. □

We prove that in all six open cases the same-length splicing languages are context-free valence languages over $\mathbb{Z}^1$ and thus context-sensitive.

**Theorem 7.** *$S_{sl}(\mathsf{REG}, \mathsf{CF}) \subseteq \mathsf{CF}(\mathbb{Z}^1)$ and $S_{sl}(\mathsf{CF}, \mathsf{REG}) \subseteq \mathsf{CF}(\mathbb{Z}^1)$.*

*Proof.* Let $h = (V, L, R)$ be a splicing system. As described before, the language $C(L, R)$ codes the splicing ingredients in the free case, and can be used to obtain upper bounds. For the (REG,CF) and (CF,REG) types of splicing we have argued that $C(L, R)$ is a context-free language, see Lemma 2.

To extend the equality $\sigma_f(h) = \text{join}(C(L, R))$ to same-length splicing we restrict $C(L, R)$ to pairs of initial strings having the same length: $\sigma_{sl}(h) = \text{join}(C(L, R) \cap D_{sl})$, where $D_{sl} = \{\ x_1\#y_1\$x_2\#y_2 \mid x_1, y_1, x_2, y_2 \in \Sigma^*, |x_1y_1| = |x_2y_2|\ \}$.

Note that $D_{sl}$ can be generated by a right-linear valence grammar over $\mathbb{Z}^1$, cf. Example 5(1), and consequently $C(L, R) \cap D_{sl}$ is an element of $\mathsf{CF}(\mathbb{Z}^1)$. This implies $\sigma_{sl}(h) \in \mathsf{CF}(\mathbb{Z}^1)$, as $\mathsf{CF}(\mathbb{Z}^1)$ is closed under finite state transductions, Proposition 4. □

We immediately have a minimal upper bound within the Chomsky hierarchy for the six open cases.

**Corollary 8.** (1) *$S_{sl}(\mathcal{F}_1, \mathcal{F}_2) \subseteq \mathsf{CS}$ for $\mathcal{F}_1 = \mathsf{REG}$ and $\mathcal{F}_2 = \mathsf{LIN}, \mathsf{CF}$.*
(2) *$S_{sl}(\mathcal{F}_1, \mathcal{F}_2) \subseteq \mathsf{CS}$ for $\mathcal{F}_1 = \mathsf{LIN}, \mathsf{CF}$ and $\mathcal{F}_2 = \mathsf{FIN}, \mathsf{REG}$.*

## 5 Length-increasing (decreasing) splicing

We consider length-increasing splicing (mode $\mu = in$) and length-decreasing splicing ($\mu = de$). Although the specifications of these modes are rather similar, it must be observed that their power is not always equal, for instance, $S_{in}(\mathsf{CF}, \mathsf{CF}) \subseteq \mathsf{CS}$, while $S_{de}(\mathsf{CF}, \mathsf{CF}) - \mathsf{CS} \neq \varnothing$.

In [KPS96, p.238] the question is raised whether $S_\mu(\mathsf{REG}, \mathsf{LIN})$, $\mu = in, de$, contains a non-context-free language. Our Example 1 shows there is indeed such a language for increasing mode, for decreasing mode there is a simple variant (adapting an example given in the proof of Lemma 10 in [KPS96]).

**Example 9.** Replace the initial language of Example 1 by

$$L' \ = \ cb^*a^*b^*c \ \cup \ c^*b^*c$$

and let $h' = (\{a, b, c\}, L', R)$ with $R$ as in Example 1. Now the only possible length-decreasing splicings are $(cb^ma^n \mid b^nc\ ,\ c^\ell c \mid b^mc) \vdash^{de} cb^ma^nb^mc$, where $1 + m + n < \ell + 1$ and $m + 1 < n + 1$, thus

$$\sigma_{de}(h') = \{cb^ma^nb^mc \mid m, n \geq 0 \text{ and } m < n\}$$

which is not in $\mathsf{CF}$. □

Consequently we have the following result.

**Lemma 10.** *$S_\mu(\mathsf{REG}, \mathsf{LIN}) - \mathsf{CF} \neq \varnothing$ for $\mu = in, de$.*

In fact, this already solves the minimal upper bounds for $S_{in}(\mathsf{REG}, \mathsf{LIN})$ and $S_{in}(\mathsf{REG}, \mathsf{CF})$ within the Chomsky hierarchy, as the inclusion $S_{in}(\mathsf{CS}, \mathsf{CS}) \subseteq \mathsf{CS}$ is known (Lemma 3 of [KPS96]).

For length-decreasing splicing we do not have such a convenient result. To remedy this, we prove that both $S_{de}(\mathsf{REG}, \mathsf{CF})$ and $S_{de}(\mathsf{CF}, \mathsf{REG})$ are subfamilies of $\mathsf{CF}(\mathbb{Z}^2)$, similar to the case of same-length splicing. For $S_{de}(\mathsf{REG}, \mathsf{CF})$ this answers the question whether the smallest upper bound in the Chomsky hierarchy is $\mathsf{CS}$ or $\mathsf{RE}$, whereas for $S_{de}(\mathsf{CF}, \mathsf{REG})$ this improves the known upper bound $\mathsf{CS}$. The argumentation works for increasing mode as well, so also in that case we obtain improved upper bounds within $\mathsf{CS}$.

**Theorem 11.** *$S_\mu(\mathsf{REG}, \mathsf{CF}) \subseteq \mathsf{CF}(\mathbb{Z}^2)$ and $S_\mu(\mathsf{CF}, \mathsf{REG}) \subseteq \mathsf{CF}(\mathbb{Z}^2)$, for $\mu = in, de$.*

*Proof.* Let $h = (V, L, R)$ be a splicing system. Consider the language $C(L, R)$ constructed from $L$ and $R$ as before. It is context-free for splicing of $(\mathsf{REG}, \mathsf{CF})$ type or of $(\mathsf{CF}, \mathsf{REG})$ type, see Lemma 2. We now consider *in* splicing, argumentation for *de* splicing is completely symmetric.

It is easily seen that $\sigma_{in}(h) = \mathrm{join}(C(L, R) \cap D_{in})$, where $D_{in}$ is the language $\{\ x_1 \# y_1 \$ x_2 \# y_2 \mid x_1, y_1, x_2, y_2 \in \Sigma^*, |x_1| > |x_2| \text{ and } |y_2| > |y_1|\ \}$ from Example 5(2).

As $C(L, R)$ is context-free, and $D_{in}$ is a right-linear valence language over $\mathbb{Z}^2$, we conclude that $\sigma_{in}(h)$ is in $\mathsf{CF}(\mathbb{Z}^2)$ using closure properties of these families. □

Summarizing, we have found minimal upper bounds for both length-increasing and length-decreasing splicing within the Chomsky hierarchy.

**Corollary 12.** *For $\mu = in, de$ one has*

(1) *$S_\mu(\mathsf{REG}, \mathcal{F}_2) \subseteq \mathsf{CS}$ for $\mathcal{F}_2 = \mathsf{LIN}, \mathsf{CF}$.*
(2) *$S_\mu(\mathcal{F}_1, \mathcal{F}_2) \subseteq \mathsf{CS}$ for $\mathcal{F}_1 = \mathsf{LIN}, \mathsf{CF}$ and $\mathcal{F}_2 = \mathsf{FIN}, \mathsf{REG}$.*

## 6 Self splicing

As self splicing takes a single string as both arguments for the splicing operation, the two splicing sites as described by the rules have to be located on that same string. As there are two possible orderings for the splicing sites for the first and second argument of the operation, the splicing may be in one of the following fashions.

First, we may splice according to the fashion $(x \mid yz, xy \mid z) \vdash xz$, i.e., the first site occurs before the second site. This means that a substring is removed from the input. In particular, when the second site occurs at the end of the string it is

possible to model the quotient operation on languages, cf. Lemma 6 in [KPS96], as follows. Consider languages $L_1, L_2$, and let $L = L_1 d$, where $d$ is a new symbol. Furthermore, let $R = \{ (\lambda, yd, d, \lambda) \mid y \in L_2 \}$. Then $\sigma_{sf}(V, L, R) = \{ x \mid xy \in L_1 \text{ for some } y \in L_2 \} = L_1/L_2$. As a consequence $S_{sf}(\mathsf{LIN}, \mathsf{LIN}) - \mathsf{CS} \neq \varnothing$. In fact, this construction also works for the modes free splicing, length-decreasing splicing, and same-length splicing.

Second, we may splice according to the fashion $(xy \mid z, x \mid yz) \vdash xyyz$, i.e., the first site occurs *after* the second site. This means that a substring in the input is doubled. In particular, it is possible to model the doubling operation on languages, cf. Lemma 3 in [PRS98], as follows. Consider a language $L_1$, and let $L = cL_1d$, where $c, d$ are new symbols. Furthermore, let $R = \{ (\lambda, d, c, \lambda) \}$. Then $\sigma_{sf}(V, L, R) = c \cdot \text{double}(L_1) \cdot d = c \cdot \{ xx \mid x \in L_1 \} \cdot d$. As a consequence $S_{sf}(\mathsf{REG}, \mathsf{FIN}) - \mathsf{CF} \neq \varnothing$. In fact, this construction is a variant of Example 6, where doubling was obtained for same-length splicing.

We meet these operations of quotient and doubling later in this section, in Lemma 15 and Theorem 13.

The present state of knowledge concerning self splicing seems to be summarized by the two earlier conclusions $S_{sf}(\mathsf{LIN}, \mathsf{LIN}) \not\subseteq \mathsf{CS}$ and $S_{sf}(\mathsf{REG}, \mathsf{FIN}) \not\subseteq \mathsf{CF}$, and the inclusion $S_{sf}(\mathsf{REG}, \mathsf{REG}) \subseteq \mathsf{CS}$ ([KPS96], remark following Lemma 7). We extend this latter inclusion to a larger family of initial languages. To this end we will need deterministic two-way finite state transductions, i.e., relations realized by deterministic finite state automata with a two-way input tape and a one-way output tape. These machines are capable of writing a doubled copy of their input, which makes them suitable to simulate self splicing. We use $\mathsf{2DGSM}$ to denote the family of deterministic two-way finite state transductions, and, in particular, $\mathsf{2DGSM}(\mathsf{CF})$ denotes the family of languages obtained from $\mathsf{CF}$ by applying these transductions[1].

**Theorem 13.** $S_{sf}(\mathsf{CF}, \mathsf{REG}) \subseteq \mathsf{2DGSM}(\mathsf{CF})$

*Proof.* Let $h = (V, L, R)$ be a splicing system with $L \in \mathsf{CF}$ and $\mathsf{Z}(R) \in \mathsf{REG}$. As explained in the introduction to this section, $\sigma_{sf}(h)$ can be described as $L_{\mathrm{xz}} \cup L_{\mathrm{xyyz}}$, where

$$\begin{aligned} L_{\mathrm{xz}} = \{\, xz \mid xyz \in L \text{ with } u_1 \in \mathrm{Suf}(x), v_1 \in \mathrm{Pref}(yz), \\ u_2 \in \mathrm{Suf}(xy), v_2 \in \mathrm{Pref}(z) \text{ for a } (u_1, v_1, u_2, v_2) \in R \,\} \text{ and} \\ L_{\mathrm{xyyx}} = \{\, xyyz \mid xyz \in L \text{ with } u_1 \in \mathrm{Suf}(xy), v_1 \in \mathrm{Pref}(z), \\ u_2 \in \mathrm{Suf}(x), v_2 \in \mathrm{Pref}(yz) \text{ for a } (u_1, v_1, u_2, v_2) \in R \,\} \end{aligned}$$

Let 1 and 2 be symbols not in $V$; we will use these symbols to mark the two splicing sites in a string, similarly to the symbol # in the strings of $C(L, R)$. So, let

$$L_{12} = \{\, x1y2z \mid xyz \in L \text{ with } u_1 \in \mathrm{Suf}(x), v_1 \in \mathrm{Pref}(yz),$$

[1] A generalized sequential machine, GSM, differs from a finite state transducer in that it is not allowed to read $\lambda$. For two-way machines the two notions are equivalent.

$$u_2 \in \mathrm{Suf}(xy), v_2 \in \mathrm{Pref}(z) \text{ for a } (u_1, v_1, u_2, v_2) \in R \,\}$$ and

$$L_{21} = \{\, x2y1z \mid xyz \in L \text{ with } u_1 \in \mathrm{Suf}(xy), v_1 \in \mathrm{Pref}(z),$$
$$u_2 \in \mathrm{Suf}(x), v_2 \in \mathrm{Pref}(yz) \text{ for a } (u_1, v_1, u_2, v_2) \in R \,\}$$

Note that $L_{12} \cup L_{21}$ can be obtained from $L$ (and $R$) by a (nondeterministic, one-way) finite state transduction based on $\mathsf{Z}(R)$. The transducer has to search for the two cutting points simultaneously, because the splicing sites $u_1v_1$ and $u_2v_2$ can overlap and the transducer is not allowed to go back on its input.

As $\mathsf{CF}$ is closed under finite state transductions, $L_{12} \cup L_{21}$ belongs to $\mathsf{CF}$.

It is now straightforward to design a (deterministic) two-way finite state transduction that maps $L_{12} \cup L_{21}$ onto $L_{\mathrm{xz}} \cup L_{\mathrm{xyyz}}$, as follows. It copies its input from left to right until it arrives at the symbol 1. At that moment the machine moves to the symbol 2 without copying (forward or backward depending on whether the 2 was encountered before finding the 1). At the symbol 2 it resumes a left to right scan copying the input, stopping at the right end of the tape. □

As stated in [EY71, Theorem 3.4], $\mathsf{2DGSM(CF)}$ is strictly contained in the family of context-sensitive languages. Hence we obtain an upper bound as required.

**Corollary 14.** *$S_{sf}(\mathcal{F}_1, \mathcal{F}_2) \subseteq \mathsf{CS}$ for $\mathcal{F}_1 = \mathsf{LIN}, \mathsf{CF}$ and $\mathcal{F}_2 = \mathsf{FIN}, \mathsf{REG}$.*

The construction for doubling discussed in the introduction of this section implies that also self splicings of $(\mathsf{REG}, \mathsf{LIN})$ type can yield languages outside $\mathsf{CF}$. We show that they can even define non-context-sensitive languages, i.e., we prove that the smallest upper bound in the Chomsky hierarchy of $S_{sf}(\mathsf{REG}, \mathsf{CF})$ and $S_{sf}(\mathsf{REG}, \mathsf{LIN})$ is $\mathsf{RE}$.

For context-free rules this is relatively easy, as we can directly simulate the quotient operation on linear languages using context-free splicing of regular languages. The result then follows, as every $\mathsf{RE}$ language can be written as the quotient of two linear languages ([LLR85]).

**Lemma 15.** *Let $L_1, L_2$ be linear languages over $\Sigma$ and let 1 be a symbol not in $\Sigma$. Then $1L_1/L_2 \in S_{sf}(\mathsf{REG}, \mathsf{CF})$.*

*Proof.* Assume that $L_1$ and $L_2$ are linear languages with $L_1, L_2 \subseteq \Sigma^*$ for some alphabet $\Sigma$. Let $1, 2, 3 \notin \Sigma$ be new symbols, and define $h = (\Sigma \cup \{1, 2, 3\}, L, R)$ by

$$L = 1\,\Sigma^* 2\,\Sigma^* 3$$
$$R = \{\, (1\,u, 2\,v\,3, 2\,w\,3, \lambda) \mid uv \in L_1, w \in L_2 \}$$

Since $\mathsf{LIN}$ is closed under concatenation with symbols and under shuffle with strings (but not under concatenation) we have $\mathsf{Z}(R) = \{1u\#2v3\$ \mid uv \in L_1\} \cdot \{2w3\# \mid w \in L_2\} \in \mathsf{LIN} \cdot \mathsf{LIN} \subseteq \mathsf{CF}$.

We start by proving that $\sigma_{sf}(h) \subseteq 1L_1/L_2$. Let $x \in L$ and $(x, x) \vdash_r z$ for a rule $r = (1u, 2v3, 2w3, \lambda)$. Then, because of the form of the axioms and the first

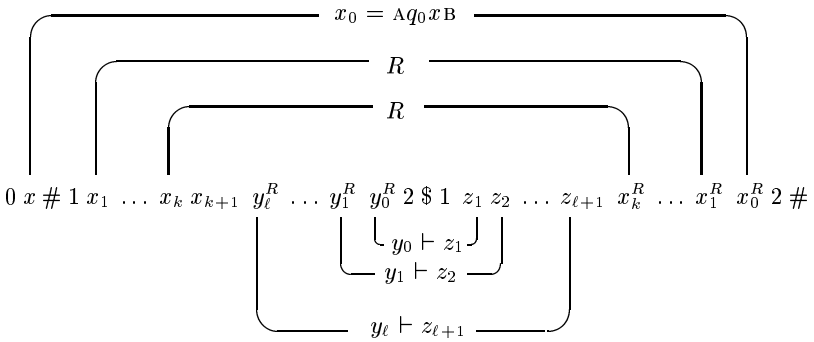



**Fig. 1.** The structure of strings in $K_{\mathcal{M}}$.

splicing site we must have $x = 1u2v3$. Moreover we have $2v3 = 2w3$, i.e., $v = w$, because we are considering self splicing. Clearly $(1u \mid 2v3\,,\ 1u2v3 \mid \lambda) \vdash_r^{sf} z = 1u \in 1L_1/L_2$, since $uv \in L_1$ and $v = w \in L_2$ by construction.

Now take $z \in L_1/L_2$, i.e., there is a $y \in L_2$ such that $zy \in L_1$. According to the definition of $h$ there is a splicing rule $r = (1z, 2y3, 2y3, \lambda) \in R$ and an axiom $1z2y3$, and so $(1z \mid 2y3\,,\ 1z2y3 \mid \lambda\,) \vdash_r 1z \in \sigma_{sf}(h)$. □

**Corollary 16.** *Let $K$ be a language over $\Sigma$ and let 1 be a symbol not in $\Sigma$. If $K \in \mathsf{RE}$, then $1K \in S_{sf}(\mathsf{REG}, \mathsf{CF})$.*

Since $\mathsf{CS}$ is closed under quotient with symbols, $1K \in \mathsf{CS}$ would imply $K \in \mathsf{CS}$. Consequently $K \in \mathsf{RE} - \mathsf{CS}$ implies $1K \in \mathsf{RE} - \mathsf{CS}$, thus the smallest upper bound in the Chomsky hierarchy of $S_{sf}(\mathsf{REG}, \mathsf{CF})$ is $\mathsf{RE}$, as formulated in the following theorem.

**Theorem 17.** $S_{sf}(\mathsf{REG}, \mathsf{CF}) - \mathsf{CS} \neq \varnothing$.

The same result holds for $S_{sf}(\mathsf{REG}, \mathsf{LIN})$, i.e., we may replace the set of rules $R$ for which $\mathsf{Z}(R) \in \mathsf{LIN} \cdot \mathsf{LIN}$ by a set $R$ with $\mathsf{Z}(R) \in \mathsf{LIN}$, cf. the proof of Lemma 15. We cannot do this directly, as in that proof. Instead, we obtain this by reconsidering the proof in [LLR85] that every recursively enumerable language is the quotient of two linear languages. The main idea is that single steps of a Turing machine can be captured by a linear grammar, provided that we represent one of the two configurations involved by its mirror image. It is also possible to represent a series of steps of the Turing machine, steps which are unrelated however, as we cannot join them into a computation without further tricks (like intersection, quotient, or ... self splicing).

We describe now our approach to code series of Turing machine computational steps. Several markers are included in the language in order to use it in the splicing process.

Let $\mathcal{M}$ be a Turing machine with state set $Q$, initial state $q_0$, final state $f$ and tape alphabet $\Gamma$. We denote the configurations of $\mathcal{M}$ by strings in $\mathrm{conf}(\mathcal{M}) =$

$\text{A}\,\Gamma^*\,Q\,\Gamma^*\,\text{B}$, where A, B are special symbols used to delimit the strings. The step relation of $\mathcal{M}$ is defined over $\text{conf}(\mathcal{M})$, and is denoted by $\vdash_{\mathcal{M}}$. We assume $\mathcal{M}$ recognizes strings $x$ over $\Sigma$ by starting in the initial configuration $\text{A}\,q_0\,x\,\text{B}$ and reaching a final configuration in $\text{A}\,\Gamma^*\,f\,\Gamma^*\,\text{B}$.

Let $0, 1, 2, \#, \$$ be symbols not in $\Sigma$. Using $w^R$ to denote the mirror image of string $w$, define the language $K_{\mathcal{M}}$ to consist of the words

$$0\,x\,\#\,1\,x_1 \ldots x_k\;x_{k+1}\;\;y_\ell^R \ldots y_1^R\,y_0^R\,2\,\$\,1\,z_1\,z_2 \ldots z_{\ell+1}\;\;x_k^R \ldots x_1^R x_0^R\,2\,\#$$

where

$x \in \Sigma^*$, $x_0 = \text{A}\,q_0\,x\,\text{B}$, $x_{k+1} \in \text{A}\,\Gamma^*\,f\,\Gamma^*\,\text{B}$,

$x_0, \ldots, x_{k+1}, y_0, \ldots, y_\ell, z_1, \ldots, z_{\ell+1} \in \text{conf}(\mathcal{M})$, for $k, \ell \geq 0$, and

$y_i \vdash_{\mathcal{M}} z_{i+1}$ for $0 \leq i \leq \ell$.

The structure of the strings in $K_{\mathcal{M}}$ is illustrated in Figure 1. A single step of a Turing machine induces just a local change in a configuration. It is an easy exercise to show that $K_{\mathcal{M}}$ is linear language, a variant of the language used in [LLR85].

**Lemma 18.** *For each Turing machine $\mathcal{M}$, $K_{\mathcal{M}} \in \mathsf{LIN}$.*

**Theorem 19.** *Let $K$ be a language over $\Sigma$ and let $0$ be a symbol not in $\Sigma$. If $K \in \mathsf{RE}$, then $0K \in S_{sf}(\mathsf{REG}, \mathsf{LIN})$.*

*Proof.* Let $K = L(\mathcal{M})$ for a deterministic Turing machine $\mathcal{M}$, and let $K_{\mathcal{M}}$ be as defined above. Now $\sigma_{sf}(h) = 0K$ for the splicing system $h = (V, L, R)$ defined by

$$\begin{aligned} V &= \Gamma \;\cup\; \{0, 1, 2\} \\ L &= 0\,\Sigma^*\,1\,(\Gamma \cup \{\text{A}, \text{B}\})^*\,2 \\ \mathsf{Z}(R) &= K_{\mathcal{M}} \end{aligned}$$

Using Lemma 18, we observe that the system $h$ is of $(\mathsf{REG}, \mathsf{LIN})$ type. The splicing rules of $h$ are of the form

$$(\;0\,x\;,\;1\,x_1 \ldots x_{k+1}\,y_\ell^R \ldots y_1^R y_0^R\,2\;,\;1\,z_1 \ldots z_{\ell+1}\,x_k^R \ldots x_1^R x_0^R\,2\;,\;\lambda\;)$$

with $x_1, \ldots, x_{k+1}, y_0, \ldots, y_\ell, z_1, \ldots, z_{\ell+1} \in \text{conf}(\mathcal{M})$, $x_0 = \text{A}\,q_0\,x\,\text{B}$ and $y_i \vdash_{\mathcal{M}} z_{i+1}$ for $0 \leq i \leq \ell$; $x_{k+1}$ is a final configuration of $\mathcal{M}$.

Because of the form of the initial strings and of the rules, the first argument of the splicing must be of the form $0\,x\,1\,x_1 \ldots x_{k+1}\,y_\ell^R \ldots y_1^R y_0^R\,2$. Since we consider self splicing, this is also the second argument. The second splicing site now enforces the equality

$$x_1 \ldots x_{k+1}\,y_\ell^R \ldots y_1^R y_0^R = z_1 \ldots z_{\ell+1}\,x_k^R \ldots x_1^R x_0^R,$$

and the marking with A and B ensures that $k = \ell$, $x_i = z_i$ for $1 \leq i \leq k+1$ and $y_j = x_j$ for $0 \leq j \leq k$. Hence $x_0 = \text{A}\,q_0\,x\,\text{B}$ is the initial configuration of $\mathcal{M}$ for the input word $x$, $x_i = y_i \vdash_{\mathcal{M}} z_{i+1} = x_{i+1}$ for $0 \leq i \leq k$, and $x_{k+1}$ is the

| | FIN | REG | LIN | CF | FIN | REG | LIN | CF | FIN | REG | LIN | CF |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $f$ | REG | REG | LIN | CF | CF | CF | RE | RE | CF | CF | RE | RE |
| $in$ | REG | REG | CF($\mathbb{Z}^2$) | CF($\mathbb{Z}^2$) | CF($\mathbb{Z}^2$) | CF($\mathbb{Z}^2$) | CS | CS | CF($\mathbb{Z}^2$) | CF($\mathbb{Z}^2$) | CS | CS |
| $de$ | REG | REG | CF($\mathbb{Z}^2$) | CF($\mathbb{Z}^2$) | CF($\mathbb{Z}^2$) | CF($\mathbb{Z}^2$) | RE | RE | CF($\mathbb{Z}^2$) | CF($\mathbb{Z}^2$) | RE | RE |
| $sl$ | LIN | LIN | CF($\mathbb{Z}^1$) | CF($\mathbb{Z}^1$) | CF($\mathbb{Z}^1$) | CF($\mathbb{Z}^1$) | RE | RE | CF($\mathbb{Z}^1$) | CF($\mathbb{Z}^1$) | RE | RE |
| $sf$ | 2(CF) | 2(CF) | RE | RE | 2(CF) | 2(CF) | RE | RE | 2(CF) | 2(CF) | RE | RE |
| | $\mathcal{F}_1 = \mathsf{REG}$ | | | | $\mathcal{F}_1 = \mathsf{LIN}$ | | | | $\mathcal{F}_1 = \mathsf{CF}$ | | | |

**Table 2.** Updated upper bounds for $S_\mu(\mathcal{F}_1, \mathcal{F}_2)$. We use 2(CF) as shorthand for 2DGSM(CF).

final configuration of $\mathcal{M}$ for $x$. Thus $x_0 \vdash_{\mathcal{M}} x_1 \vdash_{\mathcal{M}} \ldots \vdash_{\mathcal{M}} x_{k+1}$ is an accepting configuration sequence for $x$. Consequently, if $0x1x_1 \ldots x_{k+1}y_\ell^R \ldots y_1^R y_0^R 2$ splices with itself to give $0x$, then $x \in L(\mathcal{M})$.

The reverse inclusion follows along the same lines (read backwards). □

Again we obtain a negative result concerning the upper bound CS.

**Theorem 20.** $S_{sf}(\mathsf{REG}, \mathsf{LIN}) - \mathsf{CS} \neq \varnothing$.

## 7 Conclusion

We have filled the open spots in Table 1, and improved some of the known CS upper bounds given there. In Table 2 we summarize the results on the upper bounds of the four modes that we considered. Note that not all bounds given in this summary meet the original goal set in [PRS98,KPS96], to give minimal upper bounds within the Chomsky hierarchy. To get these, replace the items CF($\mathbb{Z}^1$), CF($\mathbb{Z}^2$), and 2DGSM(CF) by CS.

We now have a full insight in the complexity of the restricted splicing modes we have considered. This picture is somewhat surprising. If we order the splicing modes according to their upper bounds, we obtain different outcomes depending on the complexity of the input languages and the rules. We list a few representative examples by comparing the upper bounds in the Chomsky hierarchy.

| | |
|---|---|
| (REG, REG) | $f, in, de \prec sl \prec sf$ |
| (REG, CF) | $f \prec sl, in, de \prec sf$ |
| (CF, REG) | $f \prec sl, in, de, sf$ |
| (CF, CF) | $in \prec f, de, sl, sf$ |

The picture is more complex in the case we consider the families CF($\mathbb{Z}^k$) and 2DGSM(CF) instead of CS. We postulate here that CF($\mathbb{Z}^k$) and 2DGSM(CF) are incomparable.

Apart from the fact that self splicing seems to be the most complex operation for all types of input language and rules, it seems hard to make general observations on the relative power of restricted splicing modes. One does note that the

tables for linear and context-free initial languages coincide. However, we conjecture that, although these upper bounds are identical, the family $S_\mu(\mathsf{LIN}, \mathcal{F})$ is strictly included in $S_\mu(\mathsf{CF}, \mathcal{F})$. Similarly, we observe that for a fixed family of initial languages, the upper bounds obtained for FIN and REG rules are the same, and also the upper bounds obtained for LIN and CF rules are the same (with the exception of free splicing). For FIN and REG rules we have obtained some evidence that the families $S_\mu(\mathcal{F}, \mathsf{FIN})$ and $S_\mu(\mathcal{F}, \mathsf{REG})$ are equal for several modes of splicing, see [DHvV00] and the forthcoming thesis of the second author.

## References


[CH91] K. Culik II, T. Harju. Splicing semigroups of dominoes and DNA, *Discrete Applied Mathematics*, 31:162–177, 1991.

[DHvV00] R. Dassen, H.J. Hoogeboom, N. van Vugt. A Characterization of non-iterated splicing with regular rules. In: *Where Mathematics, Computer Science and Biology Meet* (C. Martin-Vide, V. Mitrana, eds.), Kluwer Academic Publishers, 2000, pages 319-327.

[EY71] R.W. Ehrich, S.S. Yau. Two-way sequential transductions and stack automata. *Information and Control* 18:404–446, 1971.

[FS97] H. Fernau and R. Stiebe. Regulation by valences. In: B. Rovan (ed.) Proceedings of MFCS'97, Lecture Notes in Computer Science, vol. 1295, pages 239-248. Springer-Verlag, 1997.

[FS00] H. Fernau, R. Stiebe. Sequential grammars and automata with valences. Technical Report WSI-2000-25, Wilhelm- Schickard-Institut für Informatik, Universität Tübingen, 2000. Submitted. Available via
`http://www.informatik.uni-tuebingen.de/bibliothek/wsi-reports.html`

[Gre78] S.A. Greibach. Remarks on blind and partially blind one-way multicounter machines. *Theoretical Computer Science* 7 (1978) 311- 324.

[HPP97] T. Head, Gh. Păun, D. Pixton. Language theory and molecular genetics : generative mechanisms suggested by DNA recombination. In: *Handbook of Formal Languages* (G. Rozenberg, A. Salomaa, eds.), volume 2, Springer-Verlag, 1997.

[HU79] J.E. Hopcroft, J.D. Ullman. *Introduction to automata theory, languages, and computation*, Addison-Wesley, 1979.

[HvV98] H.J. Hoogeboom, N. van Vugt. The power of H systems: does representation matter? *Computing with bio-molecules: theory and experiments* (G. Păun, ed.), Springer-Verlag, Singapore, 255–268, 1998.

[KPS96] L. Kari, G. Păun, A. Salomaa. The power of restricted splicing with rules from a regular language, *Journal of Universal Computer Science*, 2(4):224-240, 1996.

[LLR85] M. Latteux, B. Leguy, B. Ratoandromanana. The family of one-counter languages is closed under quotient, *Acta Informatica* 22:579–588, 1985.

[Pău80] G. Păun. A new generative device: valence grammars. *Revue Roumaine de Mathématiques Pures et Appliquées* 6 (1980) 911-924.

[PRS95] Gh. Păun, G. Rozenberg, A. Salomaa. Computing by splicing, *Theoretical Computer Science* 168:321–336, 1996.

[PRS96] Gh. Păun, G. Rozenberg, A. Salomaa. Restricted use of the splicing operation, *International Journal of Computer Mathematics* 60:17–32, 1996.

[PRS98] Gh. Păun, G. Rozenberg, A. Salomaa. *DNA computing. New computing paradigms*, Springer-Verlag, 1998.