Accepting splicing systems with permitting and forbidding words

Fernando Arroyo · Juan Castellanos · Jürgen Dassow · Victor Mitrana · José-Ramón Sánchez-Couso

Abstract In this paper we propose a generalization of the accepting splicing systems introduced in Mitrana et al. (Theor Comput Sci 411:2414–2422, 2010). More precisely, the input word is accepted as soon as a permitting word is obtained provided that no forbidding word has been obtained so far, otherwise it is rejected. Note that in the new variant of accepting splicing system the input word is rejected if either no permitting word is ever generated (like in Mitrana et al. in Theor Comput Sci 411:2414–2422, 2010) or a forbidding word has been generated and no permitting word had been generated before. We investigate the computational power of the new variants of accepting splicing systems and the interrelationships

F. Arroyo · J.-R. Sánchez-Couso
Department of Languages, Projects and Computer Information Systems,
University School of Informatics, Polytechnic University of Madrid,
Crta. de Valencia km. 7, 28031 Madrid, Spain
e-mail: farroyo@eui.upm.es

J.-R. Sánchez-Couso
e-mail: jcouso@eui.upm.es

J. Castellanos
Department of Artificial Intelligence, Faculty of Informatics,
Polytechnic University of Madrid, 28660 Boadilla del Monte,
Madrid, Spain
e-mail: jcastellanos@fi.upm.es

J. Dassow
Faculty of Computer Science, Otto-von-Guericke-University
of Magdeburg, P.O.Box 4120, 39016 Magdeburg, Germany
e-mail: dassow@iws.cs.uni-magdeburg.de

V. Mitrana (✉)
Department of Organization and Structure of Information,
University School of Informatics, Polytechnic University of Madrid,
Crta. de Valencia km. 7, 28031 Madrid, Spain
e-mail: victor.mitrana@upm.es

among them. We show that the new condition strictly increases the computational power of accepting splicing systems. Although there are regular languages that cannot be accepted by any of the splicing systems considered here, the new variants can accept non-regular and even non-context-free languages, a situation that is not very common in the case of (extended) finite splicing systems without additional restrictions. We also show that the smallest class of languages out of the four classes defined by accepting splicing systems is strictly included in the class of context-free languages. Solutions to a few decidability problems are immediately derived from the proof of this result.

1 Introduction

One of the basic mechanisms by which genetic material is merged is the recombination of DNA sequences under the effect of enzymatic activities. This process has been formalized as a word rewriting operation as follows: the restriction enzymes have been approximated by a finite set of rules defining the restriction sites and the DNA sequences, on which the enzymes act, have been approximated by a finite set of words usually called axioms. This is actually the main idea of the *splicing* operation viewed as a language theoretical approach of the recombinant behavior of DNA under the influence of restriction enzymes and ligases considered by T. Head in [8]. Roughly speaking, the splicing operation is applied to two DNA sequences (represented by words) which are cut at specific sites (represented by splicing rules), and the first subword of one sequence is pasted to the second segment of the other and vice versa. A new formal device to generate languages based on the iteration of splicing operation has been considered. Known as *splicing system*, this computation model has been vividly investigated in the last two decades. In spite of the vast literature devoted the topic, the real computational power of finite splicing systems is still partially unknown as the characterization of languages generated by these systems is an open problem. The problem is completely solved for extended splicing systems (a terminal alphabet is used for squeezing out the result), i.e. extended splicing systems are computationally equivalent to finite automata. Another large part of the research in this area has been focused on defining different types of splicing systems and investigating their computational power from a language generating point of view. Many variants of splicing systems have been defined and investigated; we mention here just a few of them: distributed splicing systems [4], splicing systems with multisets [6], splicing systems with permitting and forbidding contexts [7], programmed and evolving splicing systems [17]. Under certain circumstances, splicing systems are computationally complete and universal (see [18] for an overview). This result suggests the possibility to consider splicing systems as theoretical models of programmable universal DNA computers based on the splicing operation.

Several other works like [2], and the references therein, address two fundamental questions concerning splicing systems: *recognition*, which asks for an algorithm able to decide whether or not a given regular language is a splicing language, and *synthesis*, which asks for an effective procedure to construct a splicing system able to generate a given splicing language.

In [10] a novel look on splicing systems is proposed, namely splicing systems are viewed as language accepting devices and not generating ones. More precisely, a usual splicing system is used for accepting/rejecting an input word in accordance with some predefined accepting conditions. A more general version was proposed in [11] where it was called *accepting splicing system*. It is rather strange that though the theory of splicing systems is mature and well developed, an accepting model based on the splicing operation has not considered so far with two exceptions:

- The aforementioned work [10], where two well-known NP-complete problems were solved with a variant of accepting splicing systems with regular sets of splicing rules. This variant with finite sets of splicing rules was further investigated in [9].
- Work [3], where a splicing recognizer that computes by observing and contains a part exhibiting some similarity to the accepting splicing system defined in [11].

Two ways of iterating the splicing operation and two variants of accepting splicing system are investigated in [11]. Altogether, one obtains four models which are compared with each other as well as with the generating splicing systems from the computational power point of view.

This work is a continuation of [11]. While the accepting splicing systems considered in [11] reject the input word only if no word (considered as a permitting word) from a given finite set is obtained during the splicing process, in this paper we propose a similar condition for rejecting the input word. This condition is also defined by a finite set of words considered as forbidding words. More precisely, the input word is accepted as soon as a permitting word is obtained provided that no forbidding word has been obtained so far, otherwise it is rejected. Note that in the new variant of accepting splicing system the input word is rejected if either no permitting word is ever generated (like in [11]) or a forbidding word has been generated and no permitting word had been generated before. The main goal of this paper is to investigate the computational power of the new variants of splicing systems and to shed a new light on the variants introduced in [11]. Clearly, the new variants are at least as powerful as the variants considered in [11]. We actually show that the new condition strictly increases the computational power of accepting splicing systems. Although there are regular languages that cannot be accepted by any of the splicing systems considered here, the new variants can accept non-regular and even non-context-free languages, a situation that is not very common in the case of (extended) finite splicing systems without additional restrictions. Several ways of controlling the splicing process in a generating splicing systems have been considered in the literature (see, e.g., [18]) most of them leading to a maximal increase of the computational power (that of a Turing machine). This has mainly been due to the fact that the control regulates each splicing step. In another and different manner, here we deal with a condition which is checked only in the end of the computation. We also show that the smallest class of languages out of the four classes of languages defined by accepting splicing systems considered in this paper is strictly included in the class of context-free languages. Solutions to a few decidability problems are immediately derived from the proof of this result. The paper ends with a brief discussion on further directions of research in this area.

2 Basic definitions and notation

We start by summarizing the notions used throughout the paper. For all undefined notions the reader may consult [20]. An *alphabet* is a finite and nonempty set of symbols. Any finite sequence of symbols from an alphabet V is called *word* over V. The set of all words over V is denoted by V^*, the empty word is denoted by ε, and the length of the word x is denoted by $|x|$. If $w = xyz$ with x, y, z being non-empty words, then x is a prefix of w, z is a suffix of w, and y is a subword for w. Moreover, we write $\partial^l_x(w) = yz$ and $\partial^r_z(w) = xy$. By convention, if x is not a prefix (suffix) of y, then $\partial^l_x(y) = y$ ($\partial^r_x(y) = y$). Note that there is no risk of confusion with the derivative with respect to the empty word as this situation is excluded throughout this paper. For two sets of words A and B, we write $\partial^l_A(B) = \{\partial^l_x(y) \mid x \in A, y \in B\}$ and $\partial^r_B(A) = \{\partial^r_y(x) \mid x \in A, y \in B\}$. For a word x we denote by $\mathrm{Pref}_k(x)$, $\mathrm{Suff}_k(x)$ and $\mathrm{Inf}_k(x)$, the prefix, suffix and the set of subwords of x, respectively.

Note that we ignore the empty word when we define a language and the empty set when we define a class of languages.

A splicing rule over V is (following [15]) a 4-tuple $[(u_1, u_2); (u_3, u_4)]$, with $u_1, u_2, u_3, u_4 \in V^*$. For a splicing rule $r = [(u_1, u_2); (u_3, u_4)]$ and a pair of words $x, y \in V^*$, we write

$$\begin{aligned}\sigma_r(x, y) = {} & \{y_1u_3u_2x_2 \mid x = x_1u_1u_2x_2, y = y_1u_3u_4y_2\} \\ & \cup \{x_1u_1u_4y_2 \mid x = x_1u_1u_2x_2, y = y_1u_3u_4y_2\}\end{aligned}$$

for some $x_1, x_2, y_1, y_2 \in V^*$. This definition is extended to a set of splicing rules R and a language L by

$$\sigma_R(L) = \bigcup_{r \in R} \bigcup_{w_1, w_2 \in L} \sigma_r(w_1, w_2).$$

Without risk of confusion, we also denote for two languages L_1, L_2

$$\sigma_R(L_1, L_2) = \bigcup_{x_1 \in L_1} \bigcup_{x_2 \in L_2} \sigma_R(x_1, x_2), \text{ where } \sigma_R(x_1, x_2) = \bigcup_{r \in R} \sigma_r(x_1, x_2).$$

A *generating splicing system* (*GenS* for short) is a construct

$$H = (V, A, R),$$

where V is an alphabet, $A \subseteq V^*$ is the initial language, and R is a set of splicing rules over V. For a splicing system $H = (V, A, R)$ we set

$$\begin{aligned}\sigma_R^0(A) &= A, \\ \sigma_R^{i+1}(A) &= \sigma_R^i(A) \cup \sigma_R(\sigma_R^i(A)), i \geq 0, \\ \sigma_R^*(A) &= \bigcup_{i \geq 0} \sigma_R^i(A).\end{aligned} \qquad (*)$$

When the set of splicing rules is clear, we omit the subscript. Then, the language generated by H is defined as $L(H) = \sigma_R^*(A)$. Adding a terminal alphabet T we get an *extended generating splicing system* $H = (V, T, A, R)$, $T \subseteq V$, which generates the language $L(H) = T^* \cap \sigma_R^*(A)$. As all systems considered in this paper are extended systems, we shall omit the word "extended". Given a generating splicing system H as above, we say that a word $w \in L(H)$ is a *proper word* of $L(H)$, if it is generated in at least one splicing step. Clearly each word in $L(H) \setminus A$ is proper. The class of languages generated by *GenS* is denoted by $\mathcal{L}(GenS)$.

An important result in splicing theory is the so-called *Regularity Preserving Lemma* proved first in [5], as a consequence of a more general result, and then in [19] by a direct argument. It states that *GenS* with a finite set of rules and a finite initial language, i.e. A and R are both finite sets, generate exactly the class of regular languages [16]. When one allows the set of splicing rules (written as words like in [14]) to be described by regular expressions, we obtain computationally complete systems [14].

For a *GenS* $H = (V, T, A, R)$ we also introduce the following non-uniform variant of iterated splicing, where the splicing is only done with axioms. More precisely, in the non-uniform case splicing at any step occurs between a generated word in the previous stepand an axiom, differently from the general case where splicing at any step occurs between any two words generated in the previous steps. We set

$$\begin{aligned}\tau_R^0(A) &= A,\\ \tau_R^{i+1}(A) &= \sigma_R(\tau_R^i(A), A), i \geq 0, \qquad\qquad (\diamond)\\ \tau_R^*(A) &= \bigcup_{i\geq 0} \tau_R^i(A).\end{aligned}$$

The language generated by H in the non-uniform way is defined as $L_n(H) = \tau_R^*(A) \cap T^*$. The class of languages generated by $GenS$ in the non-uniform way is denoted by $\mathcal{L}_n(GenS)$.

Theorem 1 [11,16] *Both $\mathcal{L}(GenS)$ and $\mathcal{L}_n(GenS)$ equal the class of regular languages.*

We now introduce the definitions and terminology for accepting splicing systems. An *accepting splicing system* ($AccS$ for short) is a 6-tuple

$$\Gamma = (V, T, A, R, P, F),$$

where V is an alphabet, $H_\Gamma = (V, T, A, R)$ is a splicing system, while P and F are finite sets of words over V. The elements of P are called *permitting words* while those of F are called *forbidding words.*

Let $\Gamma = (V, T, A, R, P, F)$ be an $AccS$ and a word $w \in V^*$; we define the following iterated splicing that is slightly different from (*):

$$\begin{aligned}\sigma_R^0(A, w) &= \{w\},\\ \sigma_R^{i+1}(A, w) &= \sigma_R^i(A, w) \cup \sigma_R(\sigma_R^i(A, w) \cup A), i \geq 0,\\ \sigma_R^*(A, w) &= \bigcup_{i\geq 0} \sigma_R^i(A, w).\end{aligned}$$

Although this operation and that defined by (*) are denoted in the same way, there is no risk of confusion as that defined by (*) is an one-argument function while that defined here has two arguments. We say that the word $w \in T^*$ is accepted by Γ if there exists $k \geq 0$ such that

$$\begin{aligned}(i)\quad & \sigma_R^k(A, w) \cap P \neq \emptyset,\\ (ii)\quad & \sigma_R^k(A, w) \cap F = \emptyset.\end{aligned}$$

The following short discussion is in order. The reason for this definition of $\sigma_R^*(A, w)$ is two fold: on the one hand, we maintain a certain uniformity in the definitions of the two ways of acceptance by $AccS$ (see below) and on the other hand, we forbid axioms to be considered as permitting or forbidding words unless they are obtained as proper words. This restriction avoids a "funny" situation in which an $AccS$ accepts either every word whenever an axiom is a permitting word, or no word whenever an axiom is a forbidding word.

Remark 1 The following sequence of inclusions is immediate:

$$(\sigma_R^*(A \cup \{w\}) \setminus A) \subseteq \sigma_R^*(A, w) \subseteq \sigma_R^*(A \cup \{w\}).$$

On the other hand, the next equality will be useful in the sequel.

$$\begin{aligned}\sigma_R^*(A, w) = \sigma_R^*(A \cup \{w\}) \setminus \{x \in A \mid x \neq w,\\ x \text{ is not a proper word of } \sigma_R^*(A \cup \{w\})\}.\end{aligned}$$

The language accepted by an $AccS$ Γ is denoted by $L(\Gamma)$.

Fig. 1 Relationships between the classes of languages defined by accepting splicing systems

$$\begin{tikzcd}
\mathcal{L}_n(AccS) \arrow[r] \arrow[dr, dash, dotted] & \mathcal{L}(AccS) \\
\mathcal{L}_n^{\emptyset}(AccS) \arrow[u] \arrow[r] \arrow[ur] & \mathcal{L}^{\emptyset}(AccS) \arrow[u]
\end{tikzcd}$$

Remark 2 Note that every $AccS\,\Gamma = (V, T, A, R, P, F)$ with $F = \emptyset$ is actually an (extended) $AccS$ considered in [11]. This remark suggests to consider for an $AccS\,\Gamma = (V, T, A, R, P, F)$, the language $L^{\emptyset}(\Gamma) = L(\Gamma')$, where $\Gamma' = (V, T, A, R, P, \emptyset)$.

The class of languages accepted by $AccS$ and $AccS$ without forbidding words is denoted by $\mathcal{L}(AccS)$ and $\mathcal{L}^{\emptyset}(AccS)$, respectively.

For an accepting splicing system $\Gamma = (V, T, A, R, P, F)$ we also introduce the following non-uniform way of accepting words similar to the non-uniform way of generating a language by a $GenS$. The computation of such a system is nondeterministic; moreover the working mode of such a system involves words originating from the input word and a finite amount of information given by the set of axioms.

For an $AccS\,\Gamma = (V, T, A, R, P, F)$ and a word $w \in V^*$ we define the following non-uniform variant of iterated splicing, where the splicing is only done with axioms, similarly to ($\diamond$):

$$\begin{aligned}
\tau_R^0(A, w) &= \{w\},\\
\tau_R^{i+1}(A, w) &= \tau_R^i(A, w) \cup \sigma_R(\tau_R^i(A, w), A), i \geq 0,\\
\tau_R^*(A, w) &= \bigcup_{i \geq 0} \tau_R^i(A, w).
\end{aligned}$$

The language accepted by Γ in the non-uniform way is defined by:

$$L_n(\Gamma) = \{w \in T^* \mid \exists k \geq 0(\tau_R^k(A, w) \cap P \neq \emptyset) \;\&\; (\tau_R^k(A, w) \cap PF = \emptyset)\}.$$

The class of languages accepted by $AccS$ and $AccS$ without forbidding words in the non-uniform way is denoted by $\mathcal{L}_n(AccS)$ and $\mathcal{L}_n^{\emptyset}(AccS)$, respectively.

3 Computational power

The inclusions $\mathcal{L}_n^{\emptyset}(AccS) \subseteq \mathcal{L}_n(AccS)$ and $\mathcal{L}^{\emptyset}(AccS) \subseteq \mathcal{L}(AccS)$ are immediate from definitions. Furthermore, by Theorem 2 in [11] we have $\mathcal{L}_n^{\emptyset}(AccS) \subset \mathcal{L}^{\emptyset}(AccS)$. The proof of this theorem can be easily completed to a proof for the inclusion $\mathcal{L}_n(AccS) \subseteq \mathcal{L}(AccS)$. Based on these observations we now state:

Theorem 2 *The relationships indicated in Fig. 1 hold. An arrow indicates a strict inclusion while the dotted line indicates an incomparability relationship.*

Proof It suffices to provide the incomparability relationship between the families $\mathcal{L}_n(AccS)$ and $\mathcal{L}^{\emptyset}(AccS)$. We first provide a language in $\mathcal{L}_n(AccS) \setminus \mathcal{L}^{\emptyset}(AccS)$. This language, say L, is defined by the regular expression a^+b^+. It is clear that a word over $\{a, b\}$ is in L if and only if the following conditions are satisfied:

(i) it does not contain the subword ba;
(ii) it contains the subword ab.

We now construct an $AccS$ that accepts L in the non-uniform way. Let

$$\Gamma = (\{a, b, \#, \$\}, \{a, b\}, \{\#\#, \$\$\}, R, \{\$ab\$\}, \{\#ba\#\}),$$

where $R = \{[(ba, \varepsilon); (\#, \#)], [(\varepsilon, ba\#); (\#, \#)], [(ab, \varepsilon); (\$, \$)], [(\varepsilon, ab\$); (\$, \$)]\}$.

By this construction, it is easy to note that if the input word contains the subword ba, then the forbidding word $\#ba\#$ is obtained in the second splicing step. As no permitting word can be produced in the first splicing step, no matter the input word is, we infer that all input words as above are rejected by Γ. Consequently, a necessary (but not sufficient) condition for an input word to be accepted by Γ is to not contain the subword ba. On the other hand, a similar reasoning lead to the conclusion that the permitting word $\$ab\$$ is also obtained in the second splicing step provided that the input word does contain the subword ab. By these considerations, after the second step, Γ either accepts its input, provided it contains ab, but not ba, and rejects otherwise. Note that the computation of Γ on every input word containing a or b only is blocked from the first step, therefore no such word is accepted by Γ.

On the other hand, following [11], for every $AccS$ $\Gamma = (V, T, A, R, P, \emptyset)$, there exists an integer $k > 0$ such that if $w \in L(\Gamma)$, with $|w| \geq k$, then $wyw \in L(\Gamma)$ for any $y \in T^*$. In conclusion, $\{a^n b^m \mid n, m \geq 1\} \notin \mathcal{L}^{\emptyset}(AccS)$ which concludes the proof.

We now consider the regular language L' defined by the regular expression:

$$L' = (a^+(a + b)^* b^+) + (b^+(a + b)^* a^+).$$

This language lies in $\mathcal{L}^{\emptyset}(AccS)$. Indeed, the following $AccS$

$$\Gamma = (\{a, b, \#, \$\}, \{a, b\}, \{\#\#, \$\$\}, R, \{a\#\#b, b\$\$a\}, \emptyset),$$

where

$$\begin{aligned} R = \{&[(a, b); (\#, \#)], [(b, a); (\$, \$)], [(a, a\#); (\#, \#)], [(b, b\$); (\$, \$)]\} \cup \\ \{&[(\#b, b); (\#, \#)], [(\$a, a); (\$, \$)], [(a\#, \varepsilon); (\varepsilon, \#b)], [(b\$, \varepsilon); (\varepsilon, \$a)]\}. \end{aligned}$$

Assume that $L' \in \mathcal{L}_n(AccS)$ being accepted by an $AccS$ Γ; we distinguish two cases:

Case 1 There is no input word rejected by Γ because a forbidden word is obtained.

In this case as soon as a word x in $(a^+(a + b)^* b^+)$ is accepted, at least one of bx and xa is accepted as well, a contradiction.

Case 2 There are input words rejected by Γ because a forbidden word is obtained.

We take a word $w \in \{a, b\}^+$ rejected in this way; furthermore, w is rejected by a forbidding word as fast as possible, say after p splicing steps. That is, any input word that is rejected by Γ because the computation leads to a forbidden word is rejected after at least p splicing steps. Without loss of generality, we may assume that $w \in a^+(a + b)^* a^+$, the reasoning for the other case being analogous. Again we face two situations: the part of the input word leading in the fastest way to a forbidden word is a prefix or a suffix. We analyze just the situation when a suffix of w leads to a forbidden word in the fastest possible way. Let $w = uv$ and αv be the word satisfying the following two conditions: (i) it is obtained from w by the first splicing step and (ii) it leads to a forbidden word in p splicing steps. Note that bw must be accepted by Γ in less than p splicing steps.

If a suffix of bw leads to a permitting word in less than p steps, then abw is accepted by Γ which is a contradiction. If a prefix of bw leads to a permitting word in less then p steps, then a prefix of bwb does the same. As there is no possibility to reach a forbidding word in less than p steps, it follows that bwb is accepted by Γ, a contradiction.

In conclusion, $L' \notin \mathcal{L}_n(AccS)$, and we are done.

It is worth mentioning here the very simple and efficient way to reject undesired words by the first accepting splicing system from the previous proof (after two splicing steps only). This idea may be employed to show that the class $\mathcal{L}_n(AccS)$ contains "almost" all regular languages. More precisely,

Proposition 1 *For every regular language $L \subseteq V^+$ and $\ddagger \notin V$, the language $\ddagger L \in \mathcal{L}_n(AccS)$.*

Proof Let $A = (Q, V\delta, q_0, Q_f)$ be a deterministic finite automaton accepting the language L. We construct the following $AccS$:

$$\Gamma = (V \cup Q \cup \{\ddagger, \$, \#\}, V \cup \{\ddagger\}, Q\{\#\} \cup \{\#\#, \$\$\}, R, Q_f, \{\#\ddagger\#\}),$$

where

$$R = \{[(X\ddagger, \varepsilon); (\#, \#)] \mid X \in V \cup \{\ddagger\}\} \cup \{[(\varepsilon, \ddagger\#); (\#, \#)], [(\ddagger, \varepsilon); (\$, \$)]\} \cup$$
$$\{[(\$, a); (q_0, \#)] \mid a \in V\} \cup \{[(qa, \varepsilon); (\delta(q, a), \#)] \mid a \in V\}.$$

As in the proof of the previous result, after the first two consecutive steps, the forbidding word $\#\ddagger\#$ is obtained provided that the input word is of the form $x\ddagger y$ with $|x| > 0$. Therefore, all input words of this form are rejected by Γ.

Let us now analyze the computation of Γ on an input word of the form $\ddagger y$, $y \in V^*$. In the first two splicing steps, one obtains consecutively $\$y$ and then $q_0 y$. Note that the other by-product words are $\ddagger\$$ and $\$\#$ that cannot be further spliced. From now on, a word $qz, q \in Q, z \in V^*$, is computed at some step if and only if $y = xz$ and $\delta(q_0, x) = q$. In conclusion, an input word $\ddagger y$ is accepted by Γ if and only if $y \in L(A)$.

The proof is complete as soon as we note that every input word $y \in V^*$ is "inert" with respect to Γ, in the sense that no splicing can be done.

We now consider an important subclass of regular languages that can be accepted by accepting splicing For a given $k > 0$ and an alphabet V, let $S_k = (A, B, C)$ be a triple where A, B and C are sets of words over V of length k. A language L over V is called *k-locally testable in the strict sense* (k-LTSS for short) if there exists a triple $S_k = (A, B, C)$ over V as above such that for any $w \in V^*$ with $|w| \geq k$, $w \in L$ iff $[\mathrm{Pref}_k(w) \in A, \mathrm{Suff}_k(w) \in B, \mathrm{Inf}_k(w) \subseteq C]$ ([12]). When L is specified by $S_k = (A, B, C)$, we write $L = L(S_k)$. A language L is called locally testable in the strict sense (LTSS) iff L is k-LTSS for some $k > 0$. Clearly, every k-LTSS language is regular. A k-LTSS language L over V is *prefix-disjoint* if there exists a triple $S_k = (A, B, C)$ such that $L = L(S_k)$ and $\partial^l_V(L) \cap (C \cup B) = \emptyset$. A *suffix-disjoint* k-LTSS language is defined analogously.

Proposition 2 *Every prefix-disjoint or suffix-disjoint k-LTSS language belongs to $\mathcal{L}_n(AccS)$ for any $k \geq 1$.*

Proof We assume that $L = L(A, B, C)$ is a prefix-disjoint k-LTSS language over the alphabet V and $S_k = (A, B, C)$ satisfies the prefix-disjoint condition. We construct the $AccS$ $\Gamma = (U, V, I, R, P, F)$, where

- $U = V \cup \{\langle x\rangle \mid x \in A \cup B \cup C\} \cup \{¢, \$\}$,
- $R = \{[(x, \varepsilon); (\langle x\rangle, ¢)] \mid x \in A\} \cup \{[(\varepsilon, ax¢); (¢, ¢)] \mid a \in V, x \in A\} \cup$ $\{[(\langle x\rangle a, \varepsilon); (\langle y\rangle, \$)] \mid a \in V, y \in C \cup B, xa = by \text{ for some } b \in V\}$,
- $A = \{\langle x\rangle ¢ \mid x \in A\} \cup \{\langle x\rangle \$ \mid x \in C \cup B\}$,
- $P = \{\langle x\rangle \mid x \in B\}$ and $F = \{¢ax¢ \mid a \in V, x \in A\}$.

The working mode of Γ can be easily understood as soon as one makes an analogy with the construction in the proof of Proposition 1, where the words in A play the role of the marker ‡, and the symbols $\langle x \rangle$ play the role of the states.

We now prove a result which shows that the conditions considered here for an input word to be accepted increases the computational power of accepting splicing systems.

Proposition 3

1. *The class $\mathcal{L}_n(AccS)$ is incomparable with the class of regular languages.*
2. *The class $\mathcal{L}(AccS)$ is incomparable with the class of context-free languages.*

Proof As it can be easily proved that the regular language $\{a^{2n} \mid n \geq 1\}$ does not belong to $\mathcal{L}(AccS)$, it suffices to indicate a non-regular language in $\mathcal{L}_n(AccS)$ and a non-context-free language in $\mathcal{L}(AccS)$.

1. We first consider the $AccS$

$$\Gamma = (\{a, b, \$\}, \{a, b\}, \{\$\$\}, R, \{\$b\}, \{a\$\}),$$

   where

$$R = \{[(a, b); (\$, \$)], [(a, a\$); (\$, \$)], [(\$, \$); (\$b, b)]\}.$$

   We claim that

$$L_n(\Gamma) \cap a^+b^+ = \{a^m b^n \mid m > n \geq 1\}.$$

   Obviously, every word $w = a^m b^n$ with $m > n \geq 1$, is accepted by Γ. Indeed, after the first splicing step both $a^m\$$ and $\$b^n$ are obtained which tend to generate $a\$$ and $\$b$, respectively. As $m > n$, $\$b$ is firstly generated which means that w is accepted. Therefore, $\{a^m b^n \mid m > n \geq 1\} \subseteq L_n(\Gamma)$. Moreover, by the aforementioned observations, we conclude that for every word $a^m b^n$ accepted by Γ, $m > n$ must hold.
   As the language $\{a^m b^n \mid m > n \geq 1\}$ is not regular, it follows that $L_n(\Gamma)$ is not regular either.
2. We now consider the $AccS$ $\Gamma' = (V, \{a, b\}, A, R, P, F)$ defined as follows:

$$\begin{aligned} V &= \{a, b, \#, \$, \text{¢}, \ddagger, \text{¥}\}, \\ A &= \{\#\$, \$\#, \#\#, \$\text{¥}, \text{¢}\reflectbox{¢}, \ddagger\ddagger\}, \\ P &= \{a\#\#b\}, \text{ and } F = \{a\reflectbox{¢}\ddagger, \ddagger\text{¢}c, \$b\$\}, \end{aligned}$$

   and R contains the following sets of rules:

$$\begin{aligned} (i) &\quad \{[(a, b); (\$, \#)], [(b, c); (\#, \$)]\}, \\ (ii) &\quad \{[(a, a\#); (\#, \#)], [(\#c, c); (\#, \#)], [(\$b, b); (\$, \text{¥})]\}, \\ (iii) &\quad \{[(a\#, \varepsilon); (\varepsilon, \#c)]\}, \\ (iv) &\quad \{[(a, \#); (\text{¢}, \reflectbox{¢})], [(\#, c); (\text{¢}, \reflectbox{¢})], [(a\reflectbox{¢}, \varepsilon); (\ddagger, \ddagger)], [(\varepsilon, \text{¢}c); (\ddagger, \ddagger)]\}. \end{aligned}$$

We claim that $L(\Gamma') \cap \{a^n b^m c^p \mid n, m, p \geq 2\} = \{a^n b^m c^n \mid n \geq 2, m > n\}$. Let us prove first that $\{a^n b^m c^n \mid n \geq 2, m > n\} \subseteq L(\Gamma')$. A word $w = a^n b^m c^n$ with $m > n$ is accepted by Γ' as follows:

- In the first splicing step, by using the rules (i), one obtains the words: $a^n\#$, $\#c^n$, $\$b^m c^n$, and $a^n b^m\$$.

- By the rules (ii), the number of occurrences of a and c in the words of the form $a^+\#$ and $\#c^+$, respectively, is decreased simultaneously. Note that both words $a\#$ and $\#c$ are obtained for the first time in the same splicing step.
- By the rules (iii), Γ comes to taking a decision. As both words $a\#$ and $\#c$ have been generated, the permitting word $a\#\#c$ is finally obtained. So far, none of the words $a¢‡$ or $‡¢c$ has been generated. Since $m > n$, $\$b\$$ has not been generated either. In conclusion, w is accepted.

Let now $x = a^n b^m c^p$, $n, m, p \geq 2$, be a word accepted by Γ'. Note that there is only one accepting word, namely $a\#\#c$ which can be obtained by applying the splicing rule $[(a\#, \varepsilon); (\varepsilon, \#c)]$ to the pair $(a\#, \#c)$. It is plain that $a\#$ and $\#c$ is generated for the first time after n and p splicing steps, respectively, provided that the forbidden word $\$b\$$ has not been obtained yet.

If $n > p$, then $¢c$ is obtained in the $(p + 1)$-th step and the accepting word $a\#\#c$ cannot be generated earlier than the forbidden word $‡¢c$, a contradiction. The case $p > n$ leads to a similar contradiction.

It remains that $n = p$ and the pair $(a\#, \#c)$ is obtained for the first time after n splicing steps but $\$b\$$ has not been obtained after n splicing steps. This is possible if and only if $m > n$, which concludes the proof of the claim.

The language $L(\Gamma')$ is not context-free as the language $\{a^n b^m c^n \mid n \geq 2, m > n\}$ is not context-free. To this aim, one can use the Ogden's Lemma [13], where all position occupied by a and c are marked.

It is worth mentioning that despite the results presented above, the position of none of the classes $\mathcal{L}(AccS)$, $\mathcal{L}^{\emptyset}(AccS)$, $\mathcal{L}_n(AccS)$ and $\mathcal{L}_n^{\emptyset}(AccS)$ in the Chomsky hierarchy is completely known.

The smallest class among them is $\mathcal{L}_n^{\emptyset}(AccS)$. We make a first (small) step towards a solution to the previous problem by showing that this class is strictly included in the class of context-free languages.

To this aim, we need to recall a result by B.S.Baker [1]. Let $G = (N, T, S, P)$ be a phrase-structure grammar. G is said to be *terminal bounded* if each rule in P is of the form

$$x_0 A_1 x_1 A_2 x_2 \ldots x_{n-1} A_n x_n \rightarrow y_0 B_1 y_1 b_2 y_2 \ldots y_{m-1} B_m y_m,$$

where each $x_i, y_i \in T^*$, each $A_i, B_i \in N$, and either $n = 0$ or there exists $0 \leq j \leq m$ such that $|y_j| > |x_k|$ for all $1 \leq k \leq n - 1$. In other words, in each non-context-free rule of a terminal-bounded grammar, there must be some terminal word in the right side which is strictly longer than all terminal words which appear between nonterminals in the left side. In [1], it is shown that every terminal-bounded grammar generates a context-free language.

Theorem 3 *$\mathcal{L}_n^{\emptyset}(AccS)$ is strictly included in the class of context-free languages.*

Proof Let $\Gamma = (V, T, A, R, P, \emptyset)$ be an $AccS$. We define the new $AccS$ $\Gamma' = (V, T, A, R', P, \emptyset)$, where

$$R' = \{[(u, v); (r'r, ss')] \mid [(u, v); (r, s)] \in R \text{ and } r'rss' \in A\}.$$

Clearly, $L_n(\Gamma) = L_n(\Gamma')$ holds; furthermore we may assume that whenever a rule $[(u, v); (r, s)]$ is applied in a splicing step in Γ', the axiom used in this splicing step is rs.

We now construct the grammar $G = (N', T', S, Q)$, where

$$N' = \{S\} \cup (V \setminus T) \cup \{Z_a \mid a \in T\},$$
$$T' = \{¢, \$, d\} \cup T,$$

and the set Q of rules of G is defined as follows:

$$\begin{aligned}Q = &\{Y \to YZ_a \mid a \in T\} \cup \{X \to Z_a X \mid a \in T\} \cup \{Z_a \to a \mid a \in T\} \cup \\ &\{Y \to ¢, X \to \$\} \cup \bigcup_{[(u,v);(r,s)]\in R} \{¢h(rv) \to dYh(uv), h(us)\$ \to h(uv)Xd\} \cup \\ &\{S \to ¢h(x)\$ \mid x \in P\},\end{aligned}$$

where h is a morphism that is the identity on $V \setminus T$ and replaces each $a \in V \setminus T$ by Z_a. □

Claim 1 *If $y \in \tau^*_{R'}(A, w)$ and $S \Longrightarrow^* d^n ¢h(y)\$ d^m$ for some $n, m \geq 0$, then $S \Longrightarrow^* d^i ¢h(w)\$ d^j$, for some $i, j \geq 0$.*

Proof of the Claim 1 The argument is an induction on the number k of splicing steps necessary to produce y. If $k = 0$, then $y = w$ and the statement is trivially true. Let $y \in \tau^{k+1}_{R'}(A, w) = \tau^k_{R'}(A, w) \cup \sigma_{R'}(\tau^k_{R'}(A, w), A)$. If $y \in \tau^k_{R'}(A, w)$, then the statement is valid by the induction hypothesis. We analyze the case when $y \in \sigma_{R'}(\tau^k_{R'}(A, w), A)$. This means that $y \in \sigma_{R'}(z, x)$ with $z \in \tau^k_{R'}(A, w)$ and $x \in A$; more precisely the following conditions are satisfied:

(i) $z = z_1 uv z_2$ for some $z_1, z_2 \in V^*$, and $x = rs$,
(ii) either $y = z_1 us$ or $y = rv z_2$,
(iii) $[(u, v); (r, s)] \in R'$.

We consider the case $y = z_1 us$ only, the other case can be treated similarly. The derivation $S \Longrightarrow^* d^n ¢h(z_1)h(us)\$ d^m$ can be continued as follows:

$$\begin{aligned}S \Longrightarrow^* \; & d^n ¢h(z_1)h(us)\$ d^m \Longrightarrow d^n ¢h(z_1)h(uv)Xd^{m+1} \Longrightarrow^* \\ & d^n ¢h(z_1)h(uv)(z_2)\$ d^{m+1} = d^n ¢h(z)\$ d^{m+1}.\end{aligned}$$

By the induction hypothesis, we infer that $S \Longrightarrow^* d^i ¢h(w)\$ d^j$, for some $i, j \geq 0$, which concludes the proof of Claim1.

Claim 2 *If $\tau^*_{R'}(A, w) \cap P \neq \emptyset$, then $S \Longrightarrow^* d^i ¢h(w)\$ d^j$, for some $i, j \geq 0$.*

Proof of the Claim 2 The argument is based on the previous claim as soon as we note that if $y \in \tau^*_{R'}(A, w) \cap P$, then $S \Longrightarrow ¢h(y)\$$ immediately holds. Consequently, by Claim 1, $S \Longrightarrow^* d^i ¢h(w)\$ d^j$, for some $i, j \geq 0$ holds. This ends the proof of Claim 2.

As a consequence of the previous claim we may state that $L_n(\Gamma') \subseteq g(L(G))$, where g is a morphism that erases d, $¢$, $\$$ and leaves unchanged the letters in T.

We now define the following derivation in G:

$$\alpha \Rrightarrow \beta \text{ iff } \begin{cases} \alpha = S \text{ and } \beta = ¢h(x)\$, \quad \text{for some } x \in P, \\ \\ \alpha = d^n ¢h(x)\$ d^m \text{ for some } n, m \geq 0, \beta = d^i ¢h(y)\$ d^j \text{ for some } i, j \geq 0 \\ \text{ and } \alpha \Longrightarrow^* \beta \text{ uses exactly one non-context-free rule from } Q.\end{cases}$$

Naturally, $\Rrightarrow^*$ denotes the reflexive and transitive closure of he relation $\Rrightarrow$.

Claim 3 *If $S \Rrightarrow^* d^n ¢h(w)\$ d^m$ for some $n, m \geq 0$, then $\tau^*_{R'}(A, w) \cap P \neq \emptyset$.*

Proof of the Claim 3 We make use of an induction on k, the number of steps in the derivation $S \Rrightarrow^* d^n ¢h(w)\$ d^m$. If $k = 0$, the $S \Rrightarrow ¢h(w)\$$ and $w \in P$, hence $\tau^*_{R'}(A, w) \cap P \neq \emptyset$ holds.

Assume that

$$S \Rightarrow^k d^n ¢h(y)\$d^m \Rightarrow d^i ¢h(w)\$d^j. \tag{1}$$

By the induction hypothesis, we have $\tau^*_{R'}(A, y) \cap P \neq \emptyset$. Assume that the following conditions are fulfilled with respect to the derivation (1):

(a) $y = rvy_2$, $y_2 \in V^*$,
(b) $w = y_1uvy_2$, for some $y_1 \in V^*$,
(c) the non-context-free rule $¢h(rv) \to dYh(uv)$ together with several rules of the form $Y \to YZ_a$ and $Y \to ¢$ have been used in the derivation (1).

This means that y can be obtained in Γ' by splicing between w and the axiom rs, hence $\tau^*_{R'}(A, w) \cap P \neq \emptyset$. Now the proof of Claim 3 is complete.

The last claim leads to the relation $L_n(\Gamma') \supseteq g(L(G))$. As the grammar G is terminal-bounded, it follows that $L(G)$ and by the closure properties of the class of context-free languages, we conclude that $L_n(\Gamma')$ is context-free as well. □

A closer look to the previous proof suggests the following investigation. Let $\Gamma = (V, T, A, R, P, F)$ be an *AccS*; for a word $w \in T^*$ and $X \in \{P, F\}$ we define

$$X_\Gamma(w) = \begin{cases} min\{k \mid \tau^k_R(A, w) \cap X \neq \emptyset\}, & \text{if } \tau^*_R(A, w) \cap X \neq \emptyset, \\ \infty, & \text{otherwise.} \end{cases}$$

It is obvious that $w \in L_n(\Gamma)$ iff $P_\Gamma(w) < F_\Gamma(w)$ holds.

Now, returning to the proof of Theorem 3 we infer that if $P_\Gamma(x) = k$, then $d^i ¢x\$d^j \in L(G)$ for some $i + j = k$. On the other hand, if $d^i ¢x\$d^j \in L(G)$, for some $i, j \geq 0$, then $P_\Gamma(x) \leq i + j$. It is plain that given an *AccS* $\Gamma = (V, T, A, R, P, F)$ then:

- Given $x \in T^*$ both $P_\Gamma(x)$ and $F_\Gamma(x)$ are algorithmically computable.
- Given the nonnegative integer k and $X \in \{P, F\}$, the problem *Is* $\{x \in T^* \mid X_\Gamma(x) \leq k\}$ *empty/finite/infinite?* is decidable.
- The minimal k such that $X_\Gamma(x) = k$ for some $x \in T^*$, $X \in \{P, F\}$, is algorithmically computable.

From the proof of Theorem 3, the language $L_n(\Gamma, X, k) = \{x \in T^* \mid X_\Gamma(x) = k\}$, $X \in \{P, F\}$, is context-free, for every $k \geq 0$. However, this language is always regular.

Theorem 4 *Given an AccS* $\Gamma = (V, T, A, R, P, F)$*,* $X \in \{P, F\}$*, and* $k \geq 0$*, the language* $L_n(\Gamma, X, k)$ *is regular.*

Proof We give the proof for $X = P$ only. To this end, we consider the Algorithm 1, where ¢ and # are new symbols that do not belong to V: It is easy to note that

(i) $\#z\# \in E_1$ if and only if $\tau^k_R(A, z) \cap P \neq \emptyset\}$;
(ii) $\#z\# \in E_2$ if and only if $\tau^j_R(A, z) \cap P \neq \emptyset\}$, for some $j < k$.

Therefore, it follows that Algorithm 1 correctly computes $L_n(\Gamma, P, k)$.

We now take notice of the fact that the **for** loop can be accomplished by a finite transducer. A detailed construction of this transducer is left to the reader. Therefore, the **while** loop can be accomplished by a finite transducer. By the closure properties of the class of regular languages, we are done.

Algorithm 1 Procedure for computing $L_n(\Gamma, P, k)$. **Input:** $\Gamma = (V, T, A, R, P, F)$

```
1: $i := 0$;
2: $L_i := \{\#x\# \mid x \in P\}$;
3: while $i < k$ do
4:    $L_{i+1} := \emptyset$;
5:    for all $z \in L_i$ do
6:       if $z \in \{¢\}^*\{\#\}T^+\{\#\}$ then
7:          $L_{i+1} := L_{i+1} \cup \{¢z\}$;
8:          $L_{i+1} := L_{i+1} \cup \bigcup_{[(u,v);(r,s)] \in R}(\partial^r_{us\#}(\{\#\}\partial^l_{\{¢\}^*\{\#\}}(z))\{v\}V^*\{\#\} \cup \{\#\}V^*\{u\}(\partial^l_{\#rv}(\{\#\}\partial^l_{\{¢\}^*\{\#\}}(z))))$;
9:       else
10:         $L_{i+1} := L_{i+1} \cup \bigcup_{[(u,v);(r,s)] \in R}(\partial^r_{us\#}(z)\{v\}V^*\{\#\} \cup \{\#\}V^*\{u\}(\partial^l_{\#rv}(z)))$;
11:      end if
12:   end for
13:   $i := i + 1$;
14: end while
15: $E_1 := L_k \cap \{\#\}T^*\{\#\}$;
16: $E_2 := T_M(L_k)$; {$T_M$ is the translation defined by a finite transducer $M$ that deletes the whole prefix
    formed by ¢ of every input word if and only if that word is in $\{c\}^+\{\#\}T^+\{\#\}$}
17: $L_n(\Gamma, P, k) = h(E_1 \setminus E_2)$; {$h$ is a morphism that deletes the symbol #}
```

4 Final remarks

The results presented so far are intended to improve the picture concerning the computational power of the accepting models based on the splicing operation as a counterpart of the well investigated generating splicing systems. However, there is still room for improving the overall picture. For instance, the precise relationship between the class of regular languages and each of the classes $\mathcal{L}^{\emptyset}_n(AccS)$ and $\mathcal{L}^{\emptyset}(AccS)$ remains to be settled. The same for the relationships between $\mathcal{L}_n(AccS)$ and the class of context-free languages.

Another area of interest concerns the decidability properties of accepting splicing systems.

Theorem 5 *The membership problem is decidable for* $\mathcal{L}(AccS)$.

Proof Algorithm 1 solves the membership problem for an arbitrary $AccS$ $\Gamma = (V, T, A, R, P, F)$: Note that the condition in line 1 is algorithmically testable as the membership problem for $\mathcal{L}^{\emptyset}(AccS)$ is decidable (see [11]). Moreover, if the condition from line 1 is satisfied, then the algorithm eventually halts within the cycle **for**. □

Algorithm 2 Membership algorithm. **Input:** $w \in T^*$, $|w| = n$, $w \notin P$

```
1: if $w \notin L^{\emptyset}(\Gamma)$ then
2:    return false; halt;
3: else
4:    for all $k \geq 1$ do
5:       $Q := \sigma^k_R(A, w)$;
6:       if $(Q \cap F \neq \emptyset)$ then
7:          return false; halt;
8:       else
9:          if $(Q \cap P \neq \emptyset)$ then
10:            return true; halt;
11:         end if
12:      end if
13:   end for
14: end if
```

By [11], the emptiness and finiteness problems are decidable for $\mathcal{L}_n^{\emptyset}(AccS)$. The status of these problems as well as of other decision problems for the accepting splicing systems considered here is still open.

Another investigation of interest in our view is to consider the accepting splicing systems introduced here as problem solvers like in [10]. To this aim, the property of an accepting splicing systems to make a decision after a finite number of splicing steps appears to be important. In other words, the rejection of the input word is always a consequence of reaching a forbidding word. Can each accepting splicing system be equivalently transformed into an accepting splicing system having this property?

References

1. Baker, B.S.: Non-context-free grammars generating context-free languages. Inf. Control **24**, 231–246 (1974)
2. Bonizzoni, P., Mauri, G.: Regular splicing languages and subclasses. Theor. Comput. Sci. **340**, 349–363 (2005)
3. Cavaliere, M., Jonoska, N., Leupold, P.: DNA splicing: computing by observing. Nat. Comput. **8**, 157–170 (2009)
4. Csuhaj-Varjú, E., Kari, L., Păun, Gh: Test tube distributed systems based on splicing. Comput. AI **15**, 211–232 (1996)
5. Culik II, K., Harju, T.: Splicing semigroups of dominoes and DNA. Discrete Appl. Math. **31**, 261–277 (1991)
6. Denninghoff, K.L., Gatterdam, R.W.: On the undecidability of splicing systems. Intern. J. Comput. Math. **27**, 133–145 (1989)
7. Freund, R., Kari, L., Păun, Gh: DNA computing based on splicing. The existence of universal computers. Theory Comput. Syst. **32**, 69–112 (1999)
8. Head, T.: Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviours. Bull. Math. Biol. **49**, 737–759 (1987)
9. Loos, R., Malcher, A., Wotschke, D.: Descriptional complexity of splicing systems. Intern. J. Found. Comp. Sci. **19**, 813–826 (2008)
10. Loos, R., Martin-Vide, C., Mitrana, V.: Solving SAT and HPP with accepting splicing systems. In: Proceedings 9th Parallel Problem Solving from Nature (PPSN IX), LNCS 4193, pp. 771–777. Springer, Berlin (2006)
11. Mitrana, V., Petre, I., Rogojin, V.: Accepting splicing systems. Theor. Comput. Sci. **411**, 2414–2422 (2010)
12. McNaughton, R., Papert, S.: Counter-Free Automata. MIT Press, Cambridge, MA (1971)
13. Ogden, W.: A helpful result for proving inherent ambiguity. Math. Syst. Theory **2**, 191–194 (1968)
14. Păun, Gh: Regular extended H systems are computationally universal. J Autom. Lang. Comb. **1**, 27–36 (1996)
15. Păun, Gh: On the splicing operation. Discrete Appl. Math. **70**, 57–79 (1996)
16. Păun, Gh, Rozenberg, G., Salomaa, A.: Computing by splicing. Theoret. Comput. Sci. **168**, 321–336 (1996)
17. Paun, Gh, Rozenberg, G., Salomaa, A.: Computing by splicing. Programmed and evolving splicing systems. In: IEEE International Conference on Evolutionary Computing, Indianapolis, pp. 273–277 (1997)
18. Paun, Gh, Rozenberg, G., Salomaa, A.: Dna Computing—New Computing Paradigms. Springer, Berlin (1998)
19. Pixton, D.: Regularity of splicing languages. Discrete Appl. Math. **69**, 101–124 (1996)
20. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages, vol. I–III. Springer, Berlin (1997)