Site-Directed Deletion

Da-Jung Cho, Yo-Sub Han, Hwee Kim, Kai Salomaa

Site-Directed Deletion

Da-Jung Cho[1,2], Yo-Sub Han[3], Hwee Kim[4], and Kai Salomaa[5]

[1] CNRS & LSV, ENS Paris-Saclay
61, avenue du Président Wilson, CACHAN Cedex 94235, France
da-jung.cho@lsv.fr
[2] LRI, Université Paris-Sud
Bât 650, Rue Noetzlin, Gif-sur-Yvette 91190, France
dajung.cho@lri.fr
[3] Department of Computer Science, Yonsei University
50 Yonsei-Ro, Seodaemun-Gu, Seoul 03722, Republic of Korea
emmous@yonsei.ac.kr
[4] Department of Mathematics and Statistics, University of South Florida
12010 USF Cherry Drive, Tampa, FL 33620, USA
hweekim@mail.usf.edu
[5] School of Computing, Queen's University
Kingston, Ontario K7L 3N6, Canada
ksalomaa@cs.queensu.ca

Abstract. We introduce a new bio-inspired operation called a *site-directed deletion* motivated from site-directed mutagenesis performed by enzymatic activity of DNA polymerase: Given two strings x and y, a site-directed deletion partially deletes a substring of x guided by the string y that specifies which part of a substring can be deleted. We study a few decision problems with respect to the new operation and examine the closure properties of the (iterated) site-directed deletion operations. We, then, define a site-directed deletion-closed (and -free) language L and investigate its decidability properties when L is regular or context-free.

Keywords: Bio-inspired operations · Site-directed deletion · Closure properties · Decidability · Automata theory

1 Introduction

A deletion operation is one of the well-established operations [4, 5, 10, 12, 14]. We focus on a variant of deletion, *site-directed deletion mutagenesis*, which can be performed *in vitro*, whose goal is to create a deletion on a given gene sequence under enzymatic activities [19]. The activity under DNA polymerase requires a single-stranded DNA segment (called a template) and primers designated to form a partially double-stranded region with the template. In particular, primers complementarily bind to a given template excluding a region to be deleted and extend to the end of the template forming a double-stranded DNA segment.

From the formal language viewpoint, Head et al. [11] and Kari and Kopecki [15] considered a splicing system inspired by DNA recombination under enzymatic activities. Kari and Thierrin [16] modeled a contextual deletion that occurs in a context sensitive manner: It deletes a string v from a string u only if certain contexts are presented in a given set $\mathcal{C}$. Furthermore, Daley and McQuillan [3], and Enaganti et al. [6, 7] studied operations inspired by the action of DNA polymerase enzyme that plays a major role in DNA recombination. Recently, Cho et al. [2] studied the site-directed insertion operation[6] that inserts a string y at a specific position of x according to the matching outfix of x.

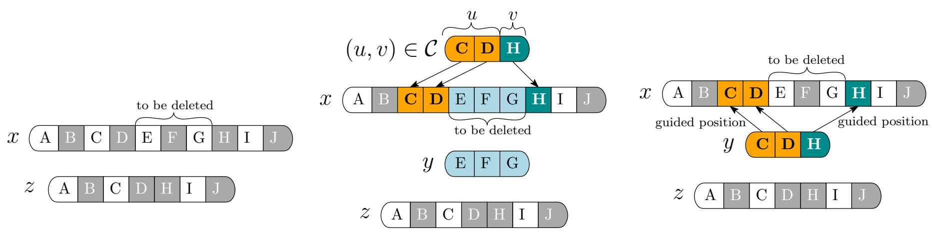
We define a new operation called *site-directed deletion* that *partially* deletes a substring from a given string x based on a guide string y that specifies which part of a substring can be deleted. In other words from a biological viewpoint, x is a template and y is a guide primer with information of the region to be deleted. We say that a site-directed deletion occurs on x if a site-directed deletion of x and y partially deletes a substring from x guided by y. This is analogous to the contextual deletion operation studied by Kari and Thierrin [16]. A contextual deletion of a string y from x occurs only if certain contexts exist in a given set $\mathcal{C}$ of contexts. Roughly speaking, contextual deletion specifies y—a string to be deleted—according to a set $\mathcal{C}$ whereas site-directed deletion deletes a substring from x according to y. Fig. 1 illustrates the difference among ordinary, contextual and site-directed deletions.

A site-directed deletion successfully generates a desired DNA sequence *in vitro* if a primer is designed to complementarily interact with a template which can result in the desired DNA sequence. From the formal language viewpoint, we consider a question of deciding whether or not the template and primer interact each other as desired. We show linear time algorithms to determine whether or not the site-directed deletion ($\mathbb{SDD}$) of two strings x and y is not empty and whether or not the site-directed deletion of x and y results in a given string z. We show that regular languages are closed under site-directed deletion, whereas context-free languages are not, and we give a context-free language L which is not closed under iterated site-directed deletion. Finally, we consider decidability for *sdd*-closedness and *sdd*-freeness on a language L.

2 Preliminaries

Let Σ be a finite alphabet of characters and Σ^* be the set of all strings over Σ. The symbol $\emptyset$ denotes the empty language and the symbol λ denotes the empty string. Given a string x, $|x|$ denotes the length of x and x^R denotes the reversal of x. A string $x = x[1]x[2] \cdots x[n] \in \Sigma^n$ is a finite sequence of n symbols. For $1 \leq i \leq j \leq n$, we denote $x[i : j] = x[i]x[i+1] \cdots x[j]$. Given a string $x = uv$, where $u, v \in \Sigma^*$, we say that u is a *prefix* of x and v is a *suffix* of x. Given a string $y = uwv$, where $u, w, v \in \Sigma^*$, we say that (u, v) is an outfix of y. If $u \neq \lambda$ and $v \neq \lambda$, then we say that (u, v) is a *non-trivial outfix* of y. Sometimes (in particular, when talking about the site-directed deletion operation) we call an outfix (u, v) simply a string uv (when it is known from the context what

[6] Site-directed insertion has earlier been considered under the name *outfix-guided insertion* [2].



(a) Ordinary deletion on x

(b) Contextual deletion of x and y with $\mathcal{C}$

(c) Site-directed deletion on x and y

Fig. 1: Examples of ordinary deletion, contextual deletion and site-directed deletion. (a) An ordinary deletion of x results in a string z removing a substring EFG. The resulting string z is one of many possible deletion outputs. (b) A contextual deletion of x and y with a set $\mathcal{C} \subseteq \Sigma^* \times \Sigma$ of contexts results in a string z if there exists an infix uyv of x, where $(u, v) \in \mathcal{C}, uv = CDH$ and $y = EFG$. (c) A site-directed deletion of two strings x and y—in other words, a template and a primer, respectively—results in a string z if $y = CDH$ is the catenation of an outfix (CD, H) of a substring $CDEFGH$ of x. The string z is the only possible site-directed deletion output when the guide string is CDH.

the components u and v are). Given a language L, we use $\overline{L}$ to denote the complement of L; $\overline{L} = \Sigma^* \setminus L$.

A nondeterministic finite automaton (NFA) is a tuple $A = (\Sigma, Q, \delta, q_0, F)$, where Σ is an input alphabet, Q is a finite set of states, $\delta \colon Q \times \Sigma \to 2^Q$ is a multivalued transition function, $q_0 \in Q$ is the initial state and $F \subseteq Q$ is a set of final states. In the usual way, δ is extended as a function $Q \times \Sigma^* \to 2^Q$ and the language recognized by A is $L(A) = \{w \in \Sigma^* \mid \delta(q_0, w) \cap F \neq \emptyset\}$. The automaton A is a deterministic finite automaton (DFA) if δ is a single valued partial function. A sequence $q_0, q_1, \ldots, q_n$ of states denotes a *path*. A string w is accepted by A if there is a labeled path from q_0 to $q_n \in F$, and we call this path an *accepting path*.

We now introduce a new operation called a *site-directed deletion* that formalizes an enzymatic reaction called the site-directed deletion mutagenesis.

Definition 1. *The site-directed deletion from a string x by y is*

$$x \overset{sdd}{\leftarrow} y = \{x_1 uvx_2 \mid x = x_1 uwvx_2, y = uv, u \neq \lambda \text{ and } v \neq \lambda\}.$$

We say that y is a deletion guide *of x if $x \overset{sdd}{\leftarrow} y \neq \emptyset$.*

We use the notation $x \overset{[y]}{\Rightarrow} z$ to indicate that z is obtained by $x \overset{sdd}{\leftarrow} y$. Note that a string x is obtained from itself by selecting the outfix to be x itself, in other words $x \overset{[x]}{\Rightarrow} x$, when $|x| \geq 2$. The site-directed deletion operation can be extended to languages as follows:

$$L_1 \overset{sdd}{\leftarrow} L_2 = \bigcup_{w_i \in L_i, i=1,2} w_1 \overset{sdd}{\leftarrow} w_2.$$

We define the *iterated site-directed deletion* on a language, which is an iterated version of the site-directed deletion operation.

Definition 2. *Site-directed deletion* $\mathbb{SDD}$ *of* L *is inductively defined as*

$$\mathbb{SDD}^{(0)}(L) = L, \text{ and } \mathbb{SDD}^{(i+1)}(L) = \mathbb{SDD}^{(i)}(L) \overset{sdd}{\leftarrow} \mathbb{SDD}^{(i)}(L), i \geq 0.$$

The iterated site-directed deletion $\mathbb{SDD}^*$ *of* L *is*

$$\mathbb{SDD}^*(L) = \bigcup_{i=0}^{\infty} \mathbb{SDD}^{(i)}(L).$$

3 Decidability of Site-Directed Deletion on Strings

We first consider the question of determining whether or not a string y can be a deletion guide of a string x; in other words, is $x \overset{sdd}{\leftarrow} y \neq \emptyset$? The site-directed deletion mutagenesis deliberately creates mutant DNA sequences using designed DNA sequences—a source DNA sequence to be partially deleted, a DNA sequence that contains the target mutant region and designed primers that indicate a starting point of synthesis. Given two strings x and y, a DNA sequence that contains a target mutant region is a deletion-guide y of a source DNA sequence x. Note that site-directed deletion cannot create a mutant DNA sequence if a DNA sequence containing a mutant region does not interact with a source DNA sequence. Thus, it is important to check whether or not designed DNA sequences can interact each other. We present a linear time algorithm that determines whether or not $x \overset{sdd}{\leftarrow} y \neq \emptyset$.

Theorem 1. *Given two strings* x *and* y*, we can determine whether or not* $x \overset{sdd}{\leftarrow} y \neq \emptyset$ *in* $O(n)$ *time, where* $|x| = n, |y| = m$ *and* $m \leq n$*.*

Proof. Our goal is to determine whether or not there exist two substrings u, v of x such that $y = uv$. In other words, we look for a pentuple (g, h, i, j, k) that satisfies the following conditions:

1. $y[1 : g] = x[j{-}g{+}1 : j]$
2. $y[m{-}h{+}1 : m] = x[k : k{+}h{-}1]$
3. $j - g + i \leq k + h - m + i$
4. $i \leq g$
5. $m - i \leq h$

Suppose there exists a pentuple (g, h, i, j, k) that satisfies all five conditions. Also suppose that there exists g' such that $y[1 : g'] = x[j-g'+1 : j]$ and $g' < g$. If $i \le g'$, then $i \le g$ and (g, h, i, j, k) also satisfies the five conditions. Therefore, for each index in x, we need to find the longest prefix of y that ends at j and the longest suffix of y that starts from k. We check the required conditions using a modified Aho-Corasick algorithm [1].

Using all prefixes of y and y^R as two dictionaries, we construct two Aho-Corasick automata $\mathcal{A}_P$ and $\mathcal{A}_S$ where the output function from a state directs to the longest matching pattern. Since we use prefixes of a string as a dictionary, it is straightforward that these modified automata can be built in $O(m)$ time using $O(m)$ space. The result is two lists of pairs (j, g), (k, h) of indices from $\mathcal{A}_P$ and $\mathcal{A}_S$, where the first element indicates the ending (starting, respectively) index and the second elements indicates the length of the longest matched pattern.

In condition 3, the inequality indicates that $m \le (g - j) + (k + h)$. From conditions 4 and 5, we know that $m \le g+h$. On the other hand, if two conditions $m \le (g - j) + (k + h)$ and $m \le g + h$ are satisfied, then we can find $i = g$ that satisfies conditions 3, 4 and 5.

We store j and k using g and h as indices respectively. Moreover, to check the condition $m \le (g - j) + (k + h)$, we only need to keep the smallest j and the largest k for same g and h, respectively. Constructing such two arrays $j[g]$ and $k[h]$ requires $O(n)$ time and $O(m)$ space. Note that we have $k[h] > k[h+1]$, since an occurrence of $y[m-h+2 : m]$ that starts at index k guarantees an occurrence of $y[m-h+1 : m]$ that starts at index $k+1$. Then we start from $g = 1$ and $h = m$, and check if $j[g] \le k[h] + 1$. If the condition is satisfied, we know that $m \le g+h$ and $m \le g - j[g] + k[h] + h$. If the condition is not satisfied, we increase g and decrease h by 1 continuously and check the condition. The runtime to compare arrays is $O(m)$, and the total runtime becomes $O(n)$. □

We next consider a problem of determining whether or not $z \in x \overset{sdd}{\leftarrow} y$ for given strings x, y and z. The biological implication of $z \in x \overset{sdd}{\leftarrow} y$ is that a site-directed deletion mutagenesis by PCR (Polymerase Chain Reaction) successfully generates a desired DNA sequence, which is denoted by z. We show that the problem can be solved in linear time.

Theorem 2. *Given three strings $x, y, z \in \Sigma^*$, we can determine whether or not $z \in x \overset{sdd}{\leftarrow} y$ in $O(n)$ time, where $n = |x|$ and $|x| \ge |z| \ge |y| \ge 2$.*

Proof. Suppose that there exist three strings $x[1 : n], y[1 : m]$ and $z[1 : l]$ such that $z \in x \overset{sdd}{\leftarrow} y$, where $n, m, l \ge 2$. Then there are decompositions of strings $x = x_1uwvx_2, y = uv$ and $z = x_1uvx_2$, where $x_1, x_2, w \in \Sigma^*$ and $u, v \ne \lambda$. We decide whether or not $z \in x \overset{sdd}{\leftarrow} y$ as follows:

1. **Scanning x and z for finding common prefix x_1u and suffix vx_2:** We scan both ends of x and z until a mismatch occurs—in other words, we find the longest matching prefix and suffix between $x[1 : n]$ and $z[1 : l]$. Let $z[1 : i] = x[1 : i]$ be the longest matching prefix, and $z[l-j+1 :$

$m] = x[n-j+1 : n]$ be the longest matching suffix, where $i, j > 1$. When $i+1 = l-j+1$, we show that the catenation of the longest matching prefix and suffix $z[1 : i] \cdot z[l-j+1 : l]$ equals to $z = x_1uvx_2$.

2. **Identifying search-range of z with respect to y**: For the case $i > l-j+1$, which implies that a suffix of $z[1 : i]$ is a prefix of $z[l-j+1 : l]$, we check an occurrence of y within $z[l-(j+m)+2 : i+m-1]$. We call the range from $l-(j+m)+2$ to $i+m-1$ a *search-range* on z. The search-range is calculated from the assumption that the longest prefix $z[1 : i]$ contains u as an infix, and the longest suffix $z[l-j+1 : l]$ contains v as an infix, where the catenation of u and v becomes y for $|u|, |v| \geq 1$. We consider a case where y matches with $z[l-j-m+2 : i+m-1]$. Suppose $y = z[l-j-m+2 : l-j+1]$. Then, $z[l-j+1 : l] = x[n-j+1 : n]$ denotes vx_2 and there exists a corresponding prefix $x[1 : l-j]$, which denotes x_1u, and we show that $z \in x \overset{sdd}{\leftarrow} y$.
3. **Finding y from search-range of z**: We can prove for all the other cases when y appears within the search-range using a similar argument. We check the existence of y within the search-range in $O(m)$ time using the KMP algorithm [17]. Thus, the total runtime is $O(n)$.

□

4 Closure Properties and Decidability on Languages

4.1 Closure Properties of Site-Directed Deletion on Languages

We show that regular languages are closed under the site-directed deletion, whereas context-free languages are not. Context-free languages are closed under site-directed deletion with regular languages. We give a context-free language L such that $\mathbb{SDD}^*(L)$ is not context-free.

Theorem 3. *If L_1 and L_2 are regular, then $L_1 \overset{sdd}{\leftarrow} L_2$ is regular.*

Proof. Let L_1 be recognized by an NFA $A = (Q_A, \Sigma, \delta, q_0, F_A)$ and L_2 be recognized by an NFA $B = (Q_B, \Sigma, \gamma, p_0, F_B)$. Let $\widetilde{Q_A} = \{\widetilde{q} \mid q \in Q_A\}$ such that Q_A is disjoint with $\widetilde{Q_A}$. We construct an NFA $C = (Q_C, \Sigma, \omega, s, F_C)$ recognizing all strings of $L_1 \overset{sdd}{\leftarrow} L_2$, where $Q_C = Q_A \times (\{\clubsuit, \heartsuit\} \cup Q_B) \cup \widetilde{Q_A} \times Q_B$, $s = (q_0, \clubsuit)$ and $F = \{(q, \heartsuit) \mid q \in F_A\} \cup \{(\widetilde{q}, p) \mid q \in F_A, p \in F_B\}$. Let α be an arbitrary symbol of Σ. We set the following transitions of ω:

(i) for $q \in Q_A, \omega((q, \clubsuit), \alpha) = \{(q', \clubsuit \mid q' \in \delta(q, \alpha)\} \cup \{(q', p') \mid q' \in \delta(q, \alpha) \text{ and } p' \in \gamma(p_0, \alpha)\}$,

(ii) for $q \in Q_A$ and $p \in Q_B$, $\omega((q, p), \alpha) \in \{(q', p') \mid q' \in \delta(q, \alpha) \text{ and } p' \in \gamma(p, \alpha)\} \cup \{(\widetilde{q''}, p') \mid q' \in \delta(q, \alpha), q'' \in \delta(q', \lambda), p' \in \gamma(p, \alpha) \text{ and } \widetilde{q} \in \widetilde{Q_A}\}$,

(iii) for $\widetilde{q} \in \widetilde{Q_A}$ and $p \in Q_B$, $\omega((\widetilde{q}, p), \alpha) \in \{(\widetilde{q'}, p') \mid q' \in \delta(q, \alpha) \text{ and } p' \in \gamma(p, \alpha)\} \cup Z_q$, where

$$Z_q = \begin{cases} \{(q', \heartsuit) \mid q' \in \delta(q, \alpha)\} & \text{if } p \in F_B, \\ \emptyset & \text{if } p \notin F_B, \end{cases}$$

(iv) for $q \in Q_A$, $\omega((q, \heartsuit), \alpha) = \{(q', \heartsuit) \mid q' \in \delta(q, \alpha)\}$.

All transitions not listed above are undefined. We show that $L(A) \overset{sdd}{\leftarrow} L(B) \subseteq L(C)$. Let $x_1uwvx_2 \in L(A)$ and $uv \in L(B)$, where $x_1, x_2, w \in \Sigma^*$ and $u, v \in \Sigma^+$. The construction is based on the idea that C uses the states $(q, \clubsuit)$ of C to process the prefix x_1, the states (q, p) to process the following substring u, the states $(\widetilde{q}, p)$ to process the substring v and the states $(q, \heartsuit)$ to process the following substring x_2. Let $\mathcal{P}_A(x)$ be an accepting path $q_0, \ldots, q_{x_1}, \ldots, q_u, \ldots, q_w, \ldots, q_u, \ldots, q_{x_2}$ of $x \in L(A)$ and $\mathcal{P}_B(y)$ be an accepting path $p_0, \ldots, p_u, \ldots, p_v$ of $y \in L(B)$. From the initial state $(q_0, \clubsuit)$ of C the transition rule (i) allows C to simulate the path $q_0, \ldots, q_{x_1}$ of $\mathcal{P}_A(x_1uwvx_2)$ reaching a state $(q_{x_1}, \clubsuit)$. When the first symbol α of u appears, the transition rule (ii) allows C to simulate both paths $q_{x_1}, \ldots, q_u$ of $\mathcal{P}_A$ and $p_0, \ldots, p_u$ of $\mathcal{P}_B$. Note that the transition (ii) allows C to nondeterministically enter the state $(\widetilde{q_w}, p_u)$ which shows that C simulates the path $q_u, \ldots, q_w$ of $\mathcal{P}_A$ reading w. Then, the transition (iii) allows C to simulate both paths $q_w, \ldots, q_v$ of $\mathcal{P}_A$ and $p_u, \ldots, p_v$ of $\mathcal{P}_B$, and C enter the state $(\widetilde{q_v}, p_v)$. Note that C enter the state $(q_v, \heartsuit)$ if $p_v \in F_B$. The transition (iv) allows C to simulate the remaining path $q_v, \ldots, q_{x_2}$ of $\mathcal{P}_A$, and simulation ends in an accepting state $(q_{x_2}, \heartsuit)$, where $q_{x_2} \in F_A$.

We now show that if C recognizes a string x_1uvx_2, then A recognizes a string x_1uwvx_2 and B recognizes a string uv, in other words; $L(C) \subseteq L(A) \overset{sdd}{\leftarrow} L(B)$. We decompose an accepting path $(q_0, \clubsuit), \ldots, (q_{x_2}, \heartsuit)$ for a string $z = x_1uvx_2 \in L(C)$ into four paths $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$ and $\mathcal{P}_4$ such that

$$\underbrace{(q_0, \clubsuit), \ldots, (q_{x_1}, \clubsuit)}_{\mathcal{P}_1}, \underbrace{(q_{x_1}, p_s), \ldots, (q_u, p_u)}_{\mathcal{P}_2},$$

$$\underbrace{(\widetilde{q_w}, p_u), \ldots, (\widetilde{q_v}, p_v)}_{\mathcal{P}_3}, \text{ and } \underbrace{(q_v, \heartsuit), \ldots, (q_{x_2}, \heartsuit)}_{\mathcal{P}_4}.$$

Path $\mathcal{P}_1$ shows that there is a path $q_0, \ldots, q_{x_2}$ in $\mathcal{P}_A$ that recognizes a string x_1. Path $\mathcal{P}_2$ shows that there exist paths $q_{x_1}, \ldots, q_u$ of $\mathcal{P}_A$ and $p_0, \ldots, p_u$ of $\mathcal{P}_B$ that recognize a string u, and path $\mathcal{P}_3$ shows that there exist paths $\widetilde{q_u}, \ldots, \widetilde{q_v}$ of $\mathcal{P}_A$ and $p_u, \ldots, p_v$ of $\mathcal{P}_B$ that recognize a string v. Note that based on the transitions (ii), C nondeterministically enters a state $(\widetilde{q}, p_u)$ from (q_u, p_u), then, C simulates a path $q, \ldots, q_w$ of $\mathcal{P}_A$ reaching $(\widetilde{q_w}, p_u)$. Path $\mathcal{P}_4$ shows that C only simulates a path $q_v, \ldots, q_{x_2}$ of $\mathcal{P}_A$, and the simulation ends on the final state $(q_{x_2}, \heartsuit)$. It implies that if C accepts a string x_1uvx_2, then there exist accepting paths on A and B that recognize strings x_1uwvx_2 and uv. □

Theorem 4. *There exist context-free languages L_1 and L_2 such that $L_1 \overset{sdd}{\leftarrow} L_2$ is not context-free.*

Proof. Consider two context-free languages $L_1 = \{a^n\$b^n\$\%c^m\# \mid n, m \geq 1\}$ and $L_2 = \{\$b^n\$c^n\# \mid n \geq 1\}$. Then, the language $(L_1 \overset{sdd}{\leftarrow} L_2) \cap (a^*\$b^*\$c^*\#) = \{a^n\$b^n\$c^n\# \mid n \geq 1\}$ is not context-free. □

The proof of Theorem 5 is omitted due to the limitation on the number of pages.

Theorem 5. *Given a context-free language L_1 and a regular language L_2, $L_1 \overset{sdd}{\leftarrow} L_2$ and $L_2 \overset{sdd}{\leftarrow} L_1$ are context-free.*

Theorem 6. *There exists a context-free language L such that $\mathbb{SDD}^*(L)$ is not context-free.*

Proof. Let $\Sigma = \{a, b, c, \$, \%, \#\}$ and define $L = L_1 \cup L_2$, where $L_1 = \{\#a^n\$b^n\$\%c^m\# \mid n, m \geq 1\}$ and $L_2 = \{\$b^n\$c^n\# \mid n \geq 1\}$. Denote $M_1 = \#a^+\$b^+\$\%c^+\#$, $M_1' = \#a^+\$b^+\$c^+\#$ and $M_2 = \$b^+\$c^+\#$ (and thus $L_1 \subseteq M_1$ and $L_2 \subseteq M_2$). We observe the following properties:

(i) It is not possible to use a string $y \in M_1 \cup M_1'$ as a deletion guide for a string $x \in M_2$ (because strings of M_2 contain only one occurrence of $\#$).
(ii) Using a string in $M_1 \cup M_1'$ as a deletion guide for a string in $M_1 \cup M_1'$ can be done only in a trivial deletion step (that produces the original deletion guide).
(iii) Using a string in M_2 as a deletion guide for a string in M_2 can be done only in a trivial deletion step.
(iv) Using a string in M_2 as a deletion guide for strings of $M_1 \cup M_1'$ always produces a string in M_1'.

We explain observation (iv) below. The first three observations follow directly from the definition of site-directed deletion.

Consider $z \in x \overset{sdd}{\leftarrow} y$ where $x \in M_1 \cup M_1'$ and $y \in M_2$. The end-marker $\#$ of y must be matched with the last symbol of x and the two occurrences of \$ in y must be matched with the two occurrences of \$ in the string x. This means that z must be the result of deleting from $x \in M_1$ the symbol % and zero or more symbols c and, if $x \in M_1'$, z must be the result of deleting from x zero or more symbols c. Not all c's can be deleted because the string $y \in M_2$ contains at least one c. Thus, in both cases $z \in M_1'$.

Since trivial deletion steps produce only the original deletion guide, observations (i)–(iv) imply that

$$L \cup \mathbb{SDD}^{(1)}(L) = L_1 \cup L_2 \cup (L_1 \overset{sdd}{\leftarrow} L_2),$$

and in particular,

$$L \cup \mathbb{SDD}^{(1)}(L) = L_1 \cup L_2 \cup \{\#a^n\$b^n\$c^n\# \mid n \geq 1\}. \tag{1}$$

The last equality is verified as in the proof of Theorem 4. Note that our current construction is a modification of the construction used in the proof of Theorem 4.

Now $L_1 \overset{sdd}{\leftarrow} L_2 \subseteq M_1'$ and none of the deletion steps (i)–(iv) produces new strings of M_2 (the steps (ii) and (iii) must use a trivial deletion step). This means that using induction on k and the observations (i)–(iv), we get that

$$\bigcup_{i=0}^{k+1} \mathbb{SDD}^{(i)}(L) = L_1 \cup L_2 \cup (\mathbb{SDD}^{(k)}(L) \overset{sdd}{\leftarrow} L_2),$$

and, thus, $\mathbb{SDD}^*(L)$ consists of L and strings obtained from L by iterated deletions that all use a string in L_2 as a guide.

When using a string $y \in L_2$ as a deletion guide for a string $x \in \{\#a^n\$b^n\$c^n\# \mid n \geq 1\}$, the only possibility is to match y with a suffix of x, because the string x has the same number of b's and c's and the string y has the same property. When y is matched with a suffix of x, the result of the deletion is just x. This means that iterating the deletions does not produce anything new and $\mathbb{SDD}^*(L) = L \cup \mathbb{SDD}^{(1)}(L)$ and from equation (1) we get that $\mathbb{SDD}^*(L) \cap \#a^+\$b^+\$c^+\# = \{\#a^n\$b^n\$c^n\# \mid n \geq 1\}$ which is not context-free. $\square$

Closure of regular languages under iterated site-directed deletion remains open.

4.2 Decidability of *sdd*-closed and *sdd*-free Languages

A language L is *sdd-closed* if $L \overset{sdd}{\leftarrow} L \subseteq L$, which implies that L does not generate a string that is not in L under the site-directed deletion operation. A language L is *sdd-free* if $x \overset{sdd}{\leftarrow} y = \emptyset$, where $x, y \in L$ and $x \neq y$. Note that a string in L is obtained only from itself by site-directed deletion if L is *sdd*-free.

We show that it is decidable whether or not a given language L is *sdd*-closed when L is regular and is undecidable when L is context-free.

Theorem 7. *Given a regular language L, it is decidable in polynomial time whether or not L is sdd-closed.*

Theorem 8. *For a given context-free language L, determining whether or not L is sdd-closed is undecidable.*

Proof. Let $((u_1, \ldots, u_n), (v_1, \ldots, v_n))$ be an instance of the *Post's Correspondence Problem* (PCP) [18], where $u_i, v_i \in \Sigma^*$ and $1 \leq i \leq n$. A solution for the instance is a sequence of integers $(i_1, \ldots, i_k)$, $i_j \in \{1, \ldots, n\}$, $j = 1, \ldots, k$ such that $u_{i_1} \cdots u_{i_k} = v_{i_1} \cdots v_{i_k}$. It is well known that the PCP is undecidable [18]. Let $\Sigma' = \Sigma \cup \{\#, \%, ¢\}$.

We consider a context-free language $L = L_1 \cup L_2$, where

$$L_1 = \{¢\$i_1 i_2 \cdots i_r \# u_{i_r}^R u_{i_{r-1}}^R \cdots u_{i_1}^R \#\%\# v_{j_1} v_{j_2} \cdots v_{j_s} \# j_s j_{s-1} \cdots j_1 \$¢\} \text{ and}$$

$$L_2 = \{\$i_1 i_2 \cdots i_r \# w \#\# w^R \# i_r i_{r-1} \cdots i_1 \$\}$$

for $w \in \Sigma^*, 1 \leq r, s,$ and $1 \leq i_r, j_s \leq n$. Then, L has two types of string

$$w_1 = ¢\$i_1 i_2 \cdots i_r \# u_{i_r}^R u_{i_{r-1}}^R \cdots u_{i_1}^R \#\%\# v_{i_1} v_{i_2} \cdots v_{i_r} \# i_r i_{r-1} \cdots i_1 \$¢ \in L_1, \text{ and}$$

$$w_2 = \$i_1 i_2 \cdots i_r \# u_{i_r}^R u_{i_{r-1}}^R \cdots u_{i_1}^R \#\# u_{i_1} u_{i_2} \cdots u_{i_r} \# i_r i_{r-1} \cdots i_1 \$ \in L_2.$$

Here, $w = u_{i_r}^R u_{i_{r-1}}^R \cdots u_{i_1}^R$ and it follows $w^R = u_{i_1} u_{i_2} \cdots u_{i_{r-1}}$. We claim that the PCP has a solution if and only if L is not *sdd*-closed, and we prove both implications of the claim.

- "If the PCP has a solution, then L is not *sdd*-closed": Suppose that $(i_1, \ldots, i_r)$ is a solution for the PCP, which implies $u_{i_1} u_{i_2} \cdots u_{i_r} = v_{i_1} v_{i_2} \cdots v_{i_r}$. We decompose $w_1 = x_1 u w v x_2$ and $w_2 = uv$, where $x_1, x_2 = ¢, w = \%$, and

$$u = \$ i_1 i_2 \cdots i_r \# u_{i_r}^R u_{i_{r-1}}^R \cdots u_{i_1}^R \#, v = \# u_{i_1} u_{i_2} \cdots u_{i_{r-1}} \# i_r i_{r-1} \cdots i_1 \$.$$

Then, the claim holds that L is not *sdd*-closed since

$$w_1 \overset{sdd}{\leftarrow} w_2 = ¢\$ i_1 i_2 \cdots i_r \# w \# \# w^R \# i_r i_{r-1} \cdots i_1 \$ ¢,$$

which is not in L—there is no string contains all occurrences of symbols $¢, \$$ and $\#$ without the symbol $\%$.
- "If the PCP has no solution, then L is *sdd*-closed": We consider two strings

$$w_1 = ¢\$ i_1 i_2 \cdots i_r \# u_{i_r}^R u_{i_{r-1}}^R \cdots u_{i_1}^R \# \% \# v_{j_1} v_{j_2} \cdots v_{j_s} \# j_s j_{s-1} \cdots j_1 \$ ¢ \in L_1 \text{ and}$$

$$w_2 = \$ i_1 i_2 \cdots i_r \# u_{i_r}^R u_{i_{r-1}}^R \cdots u_{i_1}^R \# \# u_{i_1} u_{i_2} \cdots u_{i_r} \# i_r i_{r-1} \cdots i_1 \$ \in L_2.$$

We consider the following four possible cases of $L \overset{sdd}{\leftarrow} L$, and show that there is a contradiction of the assumption.

(i) $L_1 \overset{sdd}{\leftarrow} L_1$: A string $w_1 \in L_1$ itself is a deletion guide of w_1, which is obtained by $w_1 \overset{[w_1]}{\Rightarrow} w_1$, and the claim holds that $L \overset{sdd}{\leftarrow} L \subseteq L$.

(ii) $L_2 \overset{sdd}{\leftarrow} L_2$: Similarly, a string $w_2 \in L_2$ is obtained by $w_2 \overset{[w_2]}{\Rightarrow} w_2$, and it implies that $L \overset{sdd}{\leftarrow} L \subseteq L$.

(iii) $L_2 \overset{sdd}{\leftarrow} L_1$: A string $w_1 \in L_1$ begins and ends with the symbol $¢$, which does not occur in $w_2 \in L_2$, and it also implies that $L \overset{sdd}{\leftarrow} L \subseteq L$.

(iv) $L_1 \overset{sdd}{\leftarrow} L_2$: Suppose that $w_2 \in L_2$ is a deletion guide of $w_1 \in L_1$ such that $w_1 = x_1 u w v x_2$ and $w_2 = uv$, where $u, n \neq \lambda$. Since w_1 begins and ends with $¢$, and w_2 does not contain occurrences of $¢$, in the decomposition of w_1, the first symbol $¢$ must be in x_1 and the last symbol $¢$ must be in x_2. Similarly, in the decomposition of w_1, $\%$ must be in w since w_2 does not contain $\%$. As an outfix (u, v) of a substring of w_1 should be $w_2 = uv$ when w_2 is a deletion-guide of w_1, it follows that

$$u = \$ i_1 i_2 \cdots i_r \# u_{i_r}^R u_{i_{r-1}}^R \cdots u_{i_1}^R \# \text{ and } v = \# v_{j_1} v_{j_2} \cdots v_{j_s} \# j_s j_{s-1} \cdots j_1 \$,$$

where $r = s, i_r = j_r$ and $v_{j_1} v_{j_2} \cdots v_{j_s} = u_{i_1} u_{i_2} \cdots u_{i_{r-1}} = (u_{i_r}^R u_{i_{r-1}}^R \cdots u_{i_1}^R)^R$. Thus, $(i_1, \ldots, i_r)$ is a solution for the PCP—a contradiction.

Therefore, it is undecidable whether or not L is *sdd*-closed since the PCP is undecidable. □

We also show that it is decidable whether or not L is *sdd*-free in polynomial time when L is regular, and it is undecidable when L is context-free. The insertion of y into x in an arbitrary place in x is defined as the set of strings $x \overset{ins}{\leftarrow} y = \{x_1 y x_2 \mid x = x_1 x_2 \text{ and } x_1, x_2 \in \Sigma^*\}$ [9, 13]. We use the closure property of regular languages under the insertion operation.

Lemma 1. *Consider $L \subseteq \Sigma^*$. Then L is sdd-free if and only if*

$$L \cap [\Sigma^+(L \overset{ins}{\leftarrow} \Sigma^*)\Sigma^* \cup \Sigma^*(L \overset{ins}{\leftarrow} \Sigma^+)\Sigma^* \cup \Sigma^*(L \overset{ins}{\leftarrow} \Sigma^*)\Sigma^+] = \emptyset.$$

When L is regular the language on the left-side of the equation of Lemma 1 is regular. Theorem 9 is proved using Lemma 1.

Theorem 9. *Given an NFA A, we can decide in polynomial time whether or not $L(A)$ is sdd-free.*

Theorem 10. *It is undecidable whether or not a context-free language L is sdd-free.*

We now consider the question of determining whether or not *sdd*-closed and *sdd*-free languages preserve their properties. We establish the following results which characterize *sdd*-closed and *sdd*-free languages with respect to basic operations such as intersection, union, catenation and complement. Notice that the basic operations are widely used in molecular biology, biochemistry and pharmacology, for instance, a complement under enzymatic activities inhibits an enzymatic activation that causes a disease or inflammation [8].

Theorem 11. *sdd-closed languages are closed under intersection and complement but not closed under union and catenation.*

Theorem 12. *sdd-free languages are closed under intersection but not closed under union, compliment and catenation.*

5 Conclusions

From a site-directed deletion mutagenesis under enzymatic activities, we have introduced the site-directed deletion operation. We have designed linear time algorithms for determining whether or not $x \overset{sdd}{\leftarrow} y = \emptyset$ and $z \in x \overset{sdd}{\leftarrow} y$. We have shown that $L_1 \overset{sdd}{\leftarrow} L_2$ is regular when L_1 and L_2 are regular, however, $L_1 \overset{sdd}{\leftarrow} L_2$ may not be context-free when L_1 and L_2 are context-free. Given a context-free language L_1 and a regular language L_2, both $L_1 \overset{sdd}{\leftarrow} L_2$ and $L_2 \overset{sdd}{\leftarrow} L_1$ are context-free. In addition, we have established that it is decidable whether or not a regular language L is *sdd*-closed and *sdd*-free in polynomial time, and it is undecidable whether or not a context-free language is *sdd*-closed and *sdd*-free.

Acknowledgments

We wish to thank the referees for their careful reading of our manuscript and their insightful comments and suggestions that improve the quality of the paper.

Cho was supported by Labex DigiCosme (ANR-11-LABEX-0045-DIGICOSME) operated by ANR as part of the program "Investissement d'Avenir" Idex Paris-Saclay (ANR-11-IDEX-0003-02). Han was supported by the Basic Science Research Program through NRF (2015R1D1A1A01060097) and the International

Research & Development Program of NRF (2017K1A3A1A12024971). Kim was supported in part by the NIH grant R01 GM109459. Salomaa was supported by Natural Sciences and Engineering Research Council of Canada Grant OGP0147224.

References

1. A. V. Aho and M. J. Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, 1975.
2. D.-J. Cho, Y.-S. Han, T. Ng, and K. Salomaa. Outfix-guided insertion. *Theoretical Computer Science*, 701:70–84, 2017.
3. M. Daley and I. McQuillan. Template-guided DNA recombination. *Theoretical Computer Science*, 330(2):237–250, 2005.
4. J. Dassow, V. Mitrana, and A. Salomaa. Operations and language generating devices suggested by the genome evolution. *Theoretical Computer Science*, 270(1-2):701–738, 2002.
5. M. Domaratzki. Deletion along trajectories. *Theoretical Computer Science*, 320(2-3):293–313, 2004.
6. S. K. Enaganti, O. H. Ibarra, L. Kari, and S. Kopecki. Further remarks on DNA overlap assembly. *Information and Computation*, 253, Part 1:143–154, 2017.
7. S. K. Enaganti, L. Kari, and S. Kopecki. A formal language model of DNA polymerase enzymatic activity. *Fundamenta Informaticae*, 138(1-2):179–192, 2015.
8. P. Gasque, Y. Dean, E. McGreal, J. VanBeek, and B. Morgan. Complement components of the innate immune system in health and disease in the CNS. *Immunopharmacology*, 49(1):171–186, 2000.
9. Y.-S. Han, S.-K. Ko, T. Ng, and K. Salomaa. State complexity of insertion. *International Journal of Foundations of Computer Science*, 27:863–878, 2016.
10. Y.-S. Han, S.-K. Ko, and K. Salomaa. State complexity of deletion and bipolar deletion. *Acta Informatica*, 53(1):67–85, 2016.
11. T. Head, D. Pixton, and E. Goode. Splicing systems: Regularity and below. In *Proceedings of the 8th International Workshop on DNA-Based Computers*, pages 262–268, 2002.
12. M. Ito, L. Kari, and G. Thierrin. Insertion and deletion closure of languages. *Theoretical Computer Science*, 183(1):3–19, 1997.
13. L. Kari. Insertion operations: closure properties. *Bulletin of the EATCS*, 51:181–191, 1993.
14. L. Kari. Deletion operations: closure properties. *International Journal of Computer Mathematics*, 52(1-2):23–42, 1994.
15. L. Kari and S. Kopecki. Deciding whether a regular language is generated by a splicing system. In *Proceedings of the 18th International Workshop on DNA-Based Computers*, pages 98–109, 2012.
16. L. Kari and G. Thierrin. Contextual insertions/deletions and computability. *Information and Computation*, 131(1):47–61, 1996.
17. D. E. Knuth, J. H. Morris, Jr, and V. R. Pratt. Fast pattern matching in strings. *SIAM journal on computing*, 6(2):323–350, 1977.
18. E. L. Post. A variant of a recursively unsolvable problem. *Bulletin of the American Mathematical Society*, 52(4):264–268, 1946.
19. J. Reikofski and B. Y. Tao. Polymerase chain reaction (PCR) techniques for site-directed mutagenesis. *Biotechnology Advances*, 10(4):535–547, 1992.