Site-Directed Insertion: Decision Problems, Maximality and Minimality

Da-Jung Cho, Yo-Sub Han, Kai Salomaa, Taylor J. Smith

Site-Directed Insertion: Decision Problems, Maximality and Minimality*

Da-Jung Cho[1], Yo-Sub Han[1], Kai Salomaa[2], and Taylor J. Smith[2]

[1] Department of Computer Science, Yonsei University
50, Yonsei-Ro, Seodaemun-Gu, Seoul 120-749, Republic of Korea
{ dajungcho, emmous }@yonsei.ac.kr
[2] School of Computing, Queen's University
Kingston, Ontario K7L 2N8, Canada
{ ksalomaa, tsmith }@cs.queensu.ca

Abstract. Site-directed insertion is an overlapping insertion operation that can be viewed as analogous to the overlap assembly or chop operations that concatenate strings by overlapping a suffix and a prefix of the argument strings. We consider decision problems and language equations involving site-directed insertion. By relying on the tools provided by *semantic shuffle on trajectories* we show that one variable equations involving site-directed insertion and regular constants can be solved. We consider also maximal and minimal variants of the site-directed insertion operation.

1 Introduction

Site-directed mutagenesis is one of the most important techniques for generating mutations on specific sites of DNA using polymerase chain reaction (PCR) based methods [18]. The algorithmic applications of mutagenesis have been considered e.g. by Franco and Manca [10]. Contextual insertion/deletion systems in the study of molecular computing have been used, e.g. by Kari and Thierrin [16], Daley et al. [4] and Enaganti et al. [8].

Site-directed insertion (SDI) of a string y into a string x involves matching an outfix of y with a substring of x and inserting the "middle part" of y not belonging to the outfix into x. Site-directed insertion has earlier been considered under the name *outfix-guided insertion* [2]. The operation is an overlapping variant of the insertion operation in the same sense as the overlap assembly, a.k.a. chop operation, is a variant of string concatenation [3, 9, 12, 13].

The maximal (respectively, minimal) SDI of a string y into a string x requires that, at the chosen location of x, the operation matches a maximal (respectively, minimal) outfix of y with a substring of x. This is analogous to the maximal and minimal chop operations studied by Holzer et al. [13].

* Cho and Han were supported by the Basic Science Research Program through NRF (2015R1D1A1A01060097) and the International Research & Development Program of NRF (2017K1A3A1A12024971). Salomaa and Smith were supported by Natural Sciences and Engineering Research Council of Canada Grant OGP0147224.

Site-directed insertion can be represented as a *semantic shuffle on trajectories* (SST). Shuffle on trajectories was introduced by Mateescu et al. [17] and the extension to SST is due to Domaratzki [5]. Further extensions of the shuffle-on-trajectories operation have been studied by Domaratzki et al. [7].

Here we study decision problems and language equations involving site-directed insertion and its maximal and minimal variants. The representation of SDI as a semantic shuffle on a regular set of trajectories gives regularity preserving left- and right-inverses of the operation. By the general results of Kari [15] on the decidability of equations, translated for SST by Domaratzki [5], this makes it possible to decide linear equations involving SDI where the constants are regular languages.

The maximal and minimal SDI operations do not, in general, preserve regularity. This means that the operations cannot be represented by SST [5] (on a regular set of trajectories) and the above tools are not available to deal with language equations. We show that for maximal and minimal SDI certain independence properties related to coding applications [14] can be decided in a polynomial time. The decidability of whether a regular language is closed under max/min SDI remains open.

In the last section we give a tight bound for the nondeterministic state complexity of alphabetic SDI, where the matching outfix must consist of a prefix and suffix of length exactly one. An upper bound for the state complexity of the general site-directed insertion is known but it remains open whether the bound is optimal.

2 Preliminaries

We assume the reader to be familiar with the basics of finite automata, regular languages and context-free languages [19]. Here we briefly recall some notation.

Let Σ be an alphabet and $w \in \Sigma^*$. If we can write $w = xyz$ we say that the pair (x, z) is an *outfix* of w. The outfix (x, z) is a *nontrivial outfix* of w if $x \neq \varepsilon$ and $z \neq \varepsilon$. For $L \subseteq \Sigma^*$, $\overline{L} = \Sigma^* - L$ is the complement of L.

A *nondeterministic finite automaton* (NFA) is a tuple $A = (\Sigma, Q, \delta, q_0, F)$ where Σ is the input alphabet, Q is the finite set of states, $\delta\colon Q \times \Sigma \to 2^Q$ is the transition function, $q_0 \in Q$ is the initial state and $F \subseteq Q$ is the set of final states. In the usual way δ is extended as a function $Q \times \Sigma^* \to 2^Q$ and the *language accepted 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 $|\delta(q, a)| \leq 1$ for all $q \in Q$ and $a \in \Sigma$. It is well known that the deterministic and nondeterministic finite automata recognize the class of *regular languages.*

The so called fooling set lemma gives a technique for establishing lower bounds for the size of NFAs:

Lemma 1 (Birget 1992 [1]). *Let $L \subseteq \Sigma^*$ be a regular language. Suppose that there exists a set $P = \{(x_i, w_i) \mid 1 \leq i \leq n\}$ of pairs of strings such that: (i) $x_i w_i \in L$ for $1 \leq i \leq n$, and, (ii) if $i \neq j$, then $x_i w_j \notin L$ or $x_j w_i \notin L$ for $1 \leq i, j \leq n$. Then, any minimal NFA for L has at least n states.*

Finally we recall some notions concerning operations on languages and language equations. Let $\odot$ be a binary operation on languages, and L, R are languages over an alphabet Σ.

(i) The language L is *closed under* $\odot$ if $L \odot L \subseteq L$.
(ii) The language L is $\odot$-*free* with respect to R if $L \odot R = \emptyset$.
(iii) The language L is $\odot$-*independent* with respect to R if $(L \odot \Sigma^+) \cap R = \emptyset$.
(iv) A *solution* for an equation $X \odot L = R$ (respectively, $L \odot X = R$) is a language $S \subseteq \Sigma^*$ such that $S \odot L = R$ (respectively, $L \odot S = R$).

The $\odot$-freeness and independence properties can be related to coding applications, where it might be desirable that we cannot produce new strings by applying an operation, such as site-directed insertion, to strings of the language. Domaratzki [6] defines trajectory-based codes analogously with (iii). As we will see, languages that are site-directed insertion independent with respect to themselves have a definition closely resembling outfix-codes of index one [14].

3 Site-Directed Insertion

The site-directed insertion is a partially overlapping insertion operation analogously as the overlap-assembly (or self-assembly) [3, 9] models an overlapping concatenation of strings. The overlapping concatenation operation is also called the chop operation [13].

The *site-directed insertion* (SDI) of a string y into a string x is defined as

$$x \overset{\text{sdi}}{\leftarrow} y = \{x_1 uzv x_2 \mid x = x_1 uv x_2,\ y = uzv,\ u \neq \epsilon, v \neq \epsilon\}.$$

The above definition requires that the pair (u, v) is a nontrivial outfix of the string y and uv is a substring of x. If $y = uzv$ is inserted into x by matching the outfix with a substring uv of x, we say that (u, v) is an *insertion guide* for the operation. Note that a previous paper [2] uses the name "outfix-guided insertion" for the same operation.

The site-directed insertion operation is extended in the usual way for languages by setting

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

We recall that regular languages are closed under site-directed insertion.

Proposition 1 ([2]). *If A and B are NFAs with m and n states, respectively, the language $L(A) \overset{\text{sdi}}{\leftarrow} L(B)$ has an NFA with $3mn + 2m$ states.*

A simpler form of the overlap-assembly operation requires the overlapping part of the strings to consist of a single letter. This operation is called "chop" by Holzer and Jacobi [12] but the later definition of the chop-operation [13] coincides with general overlap-assembly [9]. Analogously we define *alphabetic*

site-directed insertion by requiring that the overlapping prefix and suffix of the inserted string each consist of a single letter.

The *alphabetic site-directed insertion* of a string y into a string x is

$$x \overset{\text{a-sdi}}{\leftarrow} y = \{x_1azbx_2 \mid x = x_1abx_2,\ y = azb,\ a, b \in \Sigma,\ x_1, x_2, z \in \Sigma^*\}.$$

Note that the alphabetic site-directed insertion will have different closure properties than the standard site-directed insertion. For example, it is not difficult to see that the context-free languages are closed under alphabetic site-directed insertion, while the context-free languages are not closed under general site-directed insertion [2].

3.1 Decision problems

For a regular language L, it is decidable whether L is closed under site-directed insertion. The algorithm relies on the construction of Proposition 1 and operates in polynomial time when L is specified by a DFA [2]. Deciding whether a context-free language is closed under site-directed insertion is undecidable [2].

A language L is $\overset{\text{sdi}}{\leftarrow}$-free, or SDI-free, with respect to R if no string of R can be site-directed inserted into a string of L, that is, if $L \overset{\text{sdi}}{\leftarrow} R = \emptyset$. The language L is SDI-independent with respect to R if site-directed inserting a non-empty string into L cannot produce a string of R. Note that L being SDI-independent with respect to itself resembles the notion of L being an outfix-code of index one [14] with the difference that we require the outfix to be nontrivial. For example, $\{ab, b\}$ is SDI-independent but it is not an outfix-code of index one.

Theorem 1. *For NFAs A and B we can decide in polynomial time whether*

(i) *$L(A)$ is SDI-free (or SDI-independent) with respect to $L(B)$.*
(ii) *$L(A)$ is alphabetic SDI-free (or alphabetic SDI-independent) with respect to $L(B)$.*

For context-free languages deciding SDI-freeness and SDI-independence is undecidable.

Proposition 2. *For context-free languages L_1 and L_2 it is undecidable whether*

(i) *L_1 is SDI-free with respect to L_2,*
(ii) *L_1 is SDI-independent with respect to L_2.*

For dealing with language equations we express the site-directed insertion operation as a *semantic shuffle on a set of trajectories* (SST) due to Domaratzki [5]. The semantic shuffle extends the (syntactic) *shuffle on trajectories* originally defined by Mateescu et al. [17]. We use a simplified definition of SST that does not allow *content restriction* [5].

The *trajectory alphabet* is $\Gamma = \{0, 1, \sigma\}$ and a trajectory is a string over Γ. The semantic shuffle of $x, y \in \Sigma^*$ on a trajectory $t \in \Gamma^*$, denoted by $x \pitchfork_t y$, is defined as follows.

If $x = y = \varepsilon$, then $x \pitchfork_t y = \varepsilon$ if $t = \varepsilon$ and is undefined otherwise. If $x = ax'$, $a \in \Sigma$, $y = \varepsilon$ and $t = ct'$, $c \in \Gamma$, then

$$x \pitchfork_t \varepsilon = \begin{cases} a(x' \pitchfork_{t'} \varepsilon) \text{ if } c = 0, \\ \emptyset, \text{otherwise.} \end{cases}$$

If $x = \varepsilon$, $y = by'$, $b \in \Sigma$, and $t = ct'$, $c \in \Gamma$, then

$$\varepsilon \pitchfork_t y = \begin{cases} b(\varepsilon \pitchfork_{t'} y') \text{ if } c = 1, \\ \emptyset, \text{otherwise.} \end{cases}$$

In the case where all the strings are nonempty, for $x = ax'$, $y = by'$, $a, b \in \Sigma$, and $t = ct'$, $c \in \Gamma$, we define

$$x \pitchfork_t y = \begin{cases} a(x' \pitchfork_{t'} y) \text{ if } c = 0, \\ b(x \pitchfork_{t'} y') \text{ if } c = 1, \\ a(x' \pitchfork_{t'} y') \text{ if } a = b \text{ and } c = \sigma, \\ \emptyset, \text{ otherwise.} \end{cases}$$

Intuitively, the trajectory t is a sequence of instructions that guide the shuffle of strings x and y: 0 selects the next symbol of x, 1 the next symbol of y (these are as in the original definition of syntactic shuffle [17]) and σ represents synchronized insertion where the next symbols of the argument strings must coincide.

For $x, y \in \Sigma^*$ and $t \in \Gamma^*$, $x \pitchfork_t y$ either consists of a unique string or is undefined. For $T \subseteq \Gamma^*$, $x \pitchfork_T y = \bigcup_{t \in T} x \pitchfork_t y$ and the operation is extended in the natural way for languages over Σ.

Directly from the definition it follows that the SDI and alphabetic SDI operations can be represented as semantic shuffle on a regular set of trajectories.

Proposition 3. *Let* $T_{\text{sdi}} = 0^*\sigma^+1^*\sigma^+0^*$ *and* $T_{\text{a-sdi}} = 0^*\sigma1^*\sigma0^*$. *Then, for any languages* L_1 *and* L_2,

$$L_1 \overset{\text{sdi}}{\leftarrow} L_2 = L_1 \pitchfork_{T_{\text{sdi}}} L_2, \text{ and, } L_1 \overset{\text{a-sdi}}{\leftarrow} L_2 = L_1 \pitchfork_{T_{a-sdi}} L_2.$$

Now using the strong decidability results of Domaratzki [5] we can effectively decide linear language equations involving site-directed insertion where the constants are regular languages. The representation of SDI using SST guarantees the existence of regularity preserving left- and right-inverse of the operation. This makes it possible to use the results of Kari [15] to decide existence of solutions to linear equations where the constants are regular languages. The maximal solutions to the equations are represented using *semantic deletion along trajectories* [5]. For the deletion operation we consider a trajectory alphabet $\Delta = \{i, d, \sigma\}$. Intuitively, a trajectory $t \in \Delta^*$ guides the deletion of a string y from x as follows: a symbol i (insertion) indicates that we output the next symbol of x, a symbol d (deletion) indicates that the next symbol of y must match the next symbol of x and nothing is produced in the output and a symbol σ (synchronization) indicates that the next symbols of x and y must match and this symbol is placed in

the output. The result of deleting y from x along trajectory t is denoted $x \rightsquigarrow_t y$ and the operation is extended in the natural way for sets of trajectories and for languages.

We can express the left- and right-inverse (as defined in [15, 5]) of SDI using semantic deletion along trajectories, and these relations are used to express solutions for linear language equations. Given a binary operation $\diamond$ on strings, let $\diamond^{\text{rev}}$ be the operation defined by $x \diamond^{\text{rev}} y = y \diamond x$ for all $x, y \in \Sigma^*$. Using Theorems 6.4 and 6.5 of [5] we obtain:

Theorem 2. *Let $L, R \subseteq \Sigma^*$ be regular languages. Then for each of the following equations it is decidable whether a solution exists: (a) $X \overset{\text{sdi}}{\leftarrow} L = R$, (b) $L \overset{\text{sdi}}{\leftarrow} X = R$, (c) $X \overset{\text{a-sdi}}{\leftarrow} L = R$, (d) $L \overset{\text{a-sdi}}{\leftarrow} X = R$.*

Define $T_1 = i^*\sigma^+d^*\sigma^+i^*$, $T_1^a = i^*\sigma d^*\sigma i^*$, $T_2 = d^*\sigma^+i^*\sigma^+d^*$, and $T_2^a = d^*\sigma i^*\sigma d^*$. If a solution exists, a superset of all solutions is, respectively, for the different cases: (a) $S_a = \overline{\overline{R} \rightsquigarrow_{T_1} L}$, (b) $S_b = \overline{L(\rightsquigarrow_{T_2})^{\text{rev}}\overline{R}}$, (c) $S_c = \overline{\overline{R} \rightsquigarrow_{T_1^a} L}$, (d) $S_d = \overline{L(\rightsquigarrow_{T_2^a})^{\text{rev}}\overline{R}}$.

The above result does not give a polynomial time decision algorithm, even in the case where the languages L and R are given by DFA's. Semantic shuffle on and deletion along regular sets of trajectories preserve regularity but the operations are inherently nondeterministic and complementation blows up the size of an NFA. Note that deleting an individual string y from a string x along trajectory t is deterministic, but the automaton construction for the result of the operation on two DFA languages is nondeterministic. An explicit construction of an NFA for the syntactic shuffle of two regular languages is given in [17].

The known trajectory based methods for two variable equations [5] do not allow the trajectories to use the synchronizing symbol σ that is needed to represent the overlap of SDI. However, if we are just interested to know whether a solution exists (as opposed to finding maximal solutions), it is easy to verify that an equation $X_1 \overset{\text{sdi}}{\leftarrow} X_2 = R$ has a solution if and only if all strings of R have length at least two.

4 Maximal and minimal site-directed insertion

Holzer et al. [13] define two deterministic variants of the chop operation. The max-chop (respectively, min-chop) of strings x and y chooses the non-empty suffix of x overlapping with y to be as long (respectively, as short) as possible.

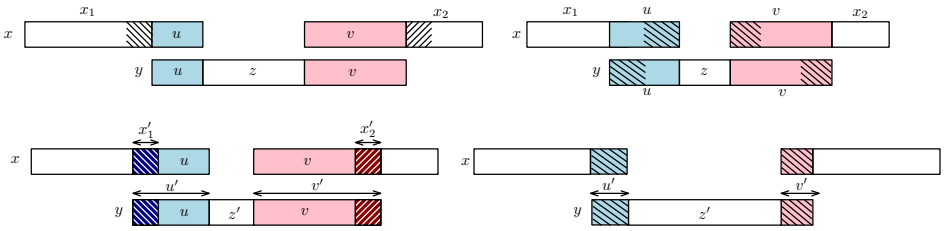
By a *maximal site-directed insertion* of string y into a string x we mean, roughly speaking, an insertion where neither the overlapping prefix nor the overlapping suffix can be properly extended. The operation is not deterministic because y could be inserted in different positions in x. At a specific position in x, a string y can be maximally (respectively, minimally) inserted in at most one way.

Formally, the *maximal site-directed insertion* (max-SDI) of a string y into string x is defined as follows:

$$x \overset{\text{max-sdi}}{\leftarrow} y = \{\, x_1uzvx_2 \mid x = x_1uvx_2,\ y = uzv,\ u \neq \epsilon, v \neq \epsilon,\ \text{and there exist no suffix } x_1' \text{ of } x_1 \text{ and prefix } x_2' \text{ of } x_2 \text{ such that } x_1'x_2' \neq \epsilon \text{ and } y = x_1'uz'vx_2',\ z' \in \Sigma^* \,\}$$

Equivalently the maximal SDI of x and y is

$$x \overset{\text{max-sdi}}{\leftarrow} y = \{\, x_1uzvx_2 \mid x = x_1uvx_2,\ y = uzv,\ u \neq \epsilon \neq v,\ \text{no suffix of } x_1u \text{ of length greater than } |u| \text{ is a prefix of } uz \text{ and no prefix of } vx_2 \text{ of length greater than } |v| \text{ is a suffix of } zv \,\}.$$



(a) Definition of maximal SDI (b) Definition of minimal SDI

Fig. 1. Insertion of y into x depicted at top left is not maximal when x and y have decompositions as depicted at bottom left.

In particular, if x and y are unary strings with $|x| \geq |y| \geq 2$, then $x \overset{\text{max-sdi}}{\leftarrow} y = x$ because the maximal overlapping outfix consists always of the entire string y. If $|x| \geq 2$ and $|y| > |x|$, then $x \overset{\text{max-sdi}}{\leftarrow} y = y$. If $|x| < 2$ or $|y| < 2$, the operation is undefined.

Example 1. Consider alphabet $\Sigma = \{a, b, c\}$. Now

$$ababab \overset{\text{max-sdi}}{\leftarrow} acbab = \{acbabab, abacbab, ababacbab\}$$

For example also the string *abacbabab* is obtained by site-directed inserting $y = acbab$ into $x = ababab$. In this operation the prefix a of y is matched with the 3rd symbol of x and the suffix b of y is matched with the 4th symbol of x. However, this operation does not satisfy the maximality condition because after the 3rd symbol of x we can match a longer suffix of y.

The *minimal site-directed insertion* (min-SDI) operation is defined as follows:

$$x \overset{\text{min-sdi}}{\leftarrow} y = \{\, x_1uzvx_2 \mid x = x_1uvx_2,\ y = uzv,\ u \neq \epsilon, v \neq \epsilon, \text{no proper nonempty suffix of } u \text{ is a prefix of } u, \text{ and no proper nonempty prefix of } v \text{ is a suffix of } v \,\}.$$

Note that in the definition of min-SDI, u and v are *unbordered words.* Figure 1 (b) illustrates the defininition of minimal SDI. The alphabetic SDI can be viewed as an "extreme" case of minimal SDI: if the first and last letter of y coincide with a substring of x of length two, then the alphabetic and minimal site-directed insertion of y in that position coincide.

If x and y are unary strings with $|x|, |y| \geq 2$, then $x \overset{\text{min-sdi}}{\leftarrow} y$ is the unary string of length $|y| + |x| - 2$ and the operation is undefined for $|x| < 2$ or $|y| < 2$.

Note that while the maximal or minimal SDI is considerably more restricted than the unrestricted SDI operation, if a string y can be site-directed inserted to a string x, it can be also maximally or minimally inserted at the same position. The result of an alphabetic insertion is always a minimal insertion. These observations are formalized in the next lemma.

Lemma 2. *Let $x, y \in \Sigma^*$.*

(i) $x \overset{\text{max-sdi}}{\leftarrow} y \subseteq x \overset{\text{sdi}}{\leftarrow} y$ *and* $x \overset{\text{a-sdi}}{\leftarrow} y \subseteq x \overset{\text{min-sdi}}{\leftarrow} y \subseteq x \overset{\text{sdi}}{\leftarrow} y$.
(ii) $x \overset{\text{sdi}}{\leftarrow} y \neq \emptyset$ *iff* $x \overset{\text{max-sdi}}{\leftarrow} y \neq \emptyset$ *iff* $x \overset{\text{min-sdi}}{\leftarrow} y \neq \emptyset$.
(iii) It is possible that $x \overset{\text{min-sdi}}{\leftarrow} y \neq \emptyset$ *and* $x \overset{\text{a-sdi}}{\leftarrow} y = \emptyset$.

Since the max-chop and min-chop operations do not preserve regularity [13], it can be expected that the same holds for maximal and minimal SDI. The proof of the following proposition is inspired by Theorem 3 of [13].

Proposition 4. *The maximal and minimal site-directed insertion do not preserve regularity.*

Proof. Let $\Sigma = \{a, b, \$, \%\}$ and choose

$$L_1 = ba^+ba^+\$, \quad L_2 = ba^+ba^+\%\$$$

We claim that

$$(L_1 \overset{\text{max-sdi}}{\leftarrow} L_2) \cap (ba^+)^3\%\$ = \{ba^mba^nba^k\%\$ \mid m \neq n \text{ or } k < n,\ m, n, k \geq 1\}$$

We denote the right side of the equation by L_{result} which is clearly nonregular. Since the strings of L_2 contain the marker % that does not occur in strings of L_1, when inserting a string $y \in L_2$ into a string of L_1 the overlapping suffix of y must consist exactly of the last symbol \$. Consider $x = ba^iba^j\$ \in L_1$ and $y = ba^rba^s\%\$ \in L_2$. In order for the resulting string to have three symbols b, a prefix of ba^r must overlap with ba^j, that is, $j \leq r$. In order for the overlap to be maximal we must have $r \neq i$ or $s < j$. These relations guarantee that the unique string in $x \overset{\text{max-sdi}}{\leftarrow} y$ is in L_{result}.

For the converse inclusion we note that, for $m \neq n$ or $k < n$,

$$ba^mba^nba^k\%\$ \in ba^mba^n\$ \overset{\text{max-sdi}}{\leftarrow} ba^nba^k\%\$.$$

For non-closure under min-SDI we claim that

$$(L_1 \overset{\text{min-sdi}}{\leftarrow} L_2) \cap (ba^+)^2\%\$ = \{ba^mba^n\%\$ \mid n > m \geq 1\} =^{\text{def}} L'_{\text{result}}.$$

Consider $x = ba^ iba^j\$ \in L_1$ and $y = ba^r ba^s\%\$ \in L_2$. In order for the result of site-directed insertion of y into x to have two b's, $ba^i ba^j$ must be a prefix of $ba^r ba^s$, that is, $i = r$ and $j \leq s$. For the site-directed insertion to be minimal, no proper non-empty prefix of $ba^i ba^j$ can be its suffix, that is $i < j$. These relations guarantee that the minimal SDI of x and y is in L'_{result}.

Conversely, for $n > m$, $ba^m ba^n\%\$ \in ba^m ba^n\$ \overset{\text{min-sdi}}{\leftarrow} ba^m ba^n\%\$$. □

In fact, extending the max-chop and min-chop constructions from Theorem 3 of [13] it would be possible to show that there exist regular languages L_1 and L_2 such that $L_1 \overset{\text{max-sdi}}{\leftarrow} L_2$ (or $L_1 \overset{\text{min-sdi}}{\leftarrow} L_2$) is not context-free. The maximal or minimal site-directed insertion of a finite language into a regular language (and vice versa) is regular.

Proposition 5. *Let R be a regular language and L a finite language. Then the languages $R \overset{\text{max-sdi}}{\leftarrow} L$, $R \overset{\text{min-sdi}}{\leftarrow} L$, $L \overset{\text{max-sdi}}{\leftarrow} R$, and $L \overset{\text{min-sdi}}{\leftarrow} R$ are regular.*

Proof. We show that $R \overset{\text{max-sdi}}{\leftarrow} L$ is regular. The other cases are very similar.

Since

$$R \overset{\text{max-sdi}}{\leftarrow} L = \bigcup_{y \in L} R \overset{\text{max-sdi}}{\leftarrow} y$$

and regular languages are closed under finite union, it is sufficient to consider the case where L consists of one string y.

Let A be an NFA for R and $y \in \Sigma^*$. We outline how an NFA B can recognize $L(A) \overset{\text{max-sdi}}{\leftarrow} y$. On an input w, B nondeterministically guesses a decomposition $w = x_1 y_1 y_2 y_3 x_2$ where $x_1 y_1 y_3 x_2 \in L(A)$, $y_1 y_2 y_3 = y$ and $y_1, y_3 \neq \varepsilon$. When reading the prefix $x_1 y_1$, B simulates a computation of A ending in a state q, then skips the substring y_2, and continues simulation of A from state q on the suffix $y_3 x_2$.

The above checks that the input is in $L(A) \overset{\text{sdi}}{\leftarrow} y$ and, additionally, B needs to verify that the insertion is maximal. This is possible because B is looking for maximal insertions of the one fixed string y.

(i) When processing the prefix x_1, the state of B remembers the last $|y| - 1$ symbols scanned. When the computation nondeterministically guesses the substrings y_1, y_2, y_3, it can then check that for no nonempty suffix x'_1 of x_1, $x'_1 y_1$ is a prefix of $y_1 y_2$. If this condition does not hold, the corresponding transition is undefined.

(ii) Similarly, when processing the (nondeterministically selected) suffix x_2 of the input, B remembers the first $|y| - 1$ symbols and is able to check that for no nonempty prefix x'_2 of x_2, $y_3 x'_2$ is a suffix of $y_2 y_3$.

If the checks in both (i) and (ii) are successful and at the end the simulation of A ends with a final state, this means that the decomposition $x_1 y_1 y_2 y_3 x_2$ gives a maximal site-directed insertion of y into a string of $L(A)$. □

4.1 Decision problems for maximal/minimal SDI

From Proposition 4 we know that the maximal or minimal SDI of regular languages need not be regular. However, for regular languages L_1 and L_2 we can decide membership in $L_1 \overset{\text{max-sdi}}{\leftarrow} L_2$ (or $L_1 \overset{\text{min-sdi}}{\leftarrow} L_2$) in polynomial time.

Lemma 3. *For DFAs A and B and $w \in \Sigma^*$ we can decide in time $O(n^6)$ whether $w \in L(A) \overset{\text{max-sdi}}{\leftarrow} L(B)$, or whether $w \in L(A) \overset{\text{min-sdi}}{\leftarrow} L(B)$.*

As we have seen, the maximal and minimal SDI operations are often more difficult to handle than the unrestricted SDI. Using Lemma 2 (ii) we note that deciding maximal (or minimal) SDI-freeness is the same as deciding SDI-freeness and by Theorem 1 we have:

Corollary 1. *For NFAs A and B we can decide in polynomial time whether or not $L(A)$ is maximal SDI-free (respectively, minimal SDI-free) with respect to $L(B)$.*

Also, deciding whether regular languages are max-SDI (or min-SDI) independent can be done in polynomial time.

Theorem 3. *For NFAs A and B, we can decide in polynomial time whether or not $L(A)$ is maximal SDI-independent (respectively, minimal SDI-independent) with respect to $L(B)$.*

Proof. Let Σ be the underlying alphabet of A and B. We verify that $L(A) \overset{\text{max-sdi}}{\leftarrow} \Sigma^+ = L(A) \overset{\text{sdi}}{\leftarrow} \Sigma^+$. The inclusion from left to right holds by Lemma 2 (i). Conversely, suppose $w \in L(A) \overset{\text{sdi}}{\leftarrow} y_1y_2y_3$, where $w = x_1y_1y_2y_3x_2$, $y_1, y_3 \neq \varepsilon$, $x_1y_1y_3x_2 \in L(A)$. Then $w \in L(A) \overset{\text{max-sdi}}{\leftarrow} x_1y_1y_2y_3x_2$, where the latter insertion uses the outfix (x_1y_1, y_3x_2) as insertion guide. The insertion is maximal because the outfix cannot be expanded. In the same way we see that $L(A) \overset{\text{min-sdi}}{\leftarrow} \Sigma^+ = L(A) \overset{\text{sdi}}{\leftarrow} \Sigma^+$. Now the claim follows by Theorem 1. □

Since the max-SDI and min-SDI operations do not preserve regularity there is no straightforward algorithm to decide whether a regular language is closed under maximal SDI or under minimal SDI. We conjecture that the problem is decidable.

Problem 1. Is there an algorithm that for a given regular language L decides whether or not $L \overset{\text{max-sdi}}{\leftarrow} L \subseteq L$ (respectively, $L \overset{\text{min-sdi}}{\leftarrow} L \subseteq L$)?

Using Proposition 5 we can decide closure of a regular language under max/min-SDI with a finite language.

Corollary 2. *Given a regular language R and a finite language F we can decide whether or not (i) $R \overset{\text{max-sdi}}{\leftarrow} F \subseteq R$, (ii) $R \overset{\text{min-sdi}}{\leftarrow} F \subseteq R$. If R is specified by a DFA and the length of the longest string in F is bounded by a constant, the algorithm works in polynomial time.*

Proof. By Proposition 5 the languages $R_{\max} = R \overset{\max-\text{sdi}}{\leftarrow} F$ and $R_{\min} = R \overset{\min-\text{sdi}}{\leftarrow} F$ are effectively regular.

Suppose $R = L(A)$ where A is a DFA with m states and underlying alphabet Σ and the length of the longest string in F is c_F. The NFA B constructed in the proof of Proposition 5 for $R_{\max}$ (or $R_{\min}$) has $O(m \cdot |\Sigma|^{c_F})$ states. Recall that the NFA stores in the state a sequence of symbols having length of the inserted string. Strictly speaking, the proof of Proposition 5 assumes that F consists of a single string, but a similar construction works for a finite language. When c_F is a constant, the size of B is polynomial in m and we can decide in polynomial time whether or not $L(B) \cap \overline{L(A)} = \emptyset$. □

The max-SDI and min-SDI operations do not preserve regularity and, consequently, they cannot be represented using semantic shuffle on trajectories. Thus, the tools developed in Section 6 of [5] to deal with language equations are not available and it remains open whether we can solve language equations involving max-SDI or min-SDI.

Problem 2. Let L and R be regular languages. Is it decidable whether the equation $X \overset{\max-\text{sdi}}{\leftarrow} L = R$ (respectively, $L \overset{\max-\text{sdi}}{\leftarrow} X = R$, $X \overset{\min-\text{sdi}}{\leftarrow} L = R$, $L \overset{\min-\text{sdi}}{\leftarrow} X = R$) has a solution?

5 Nondeterministic state complexity

The site-directed insertion (SDI) operation preserves regularity [2] (Proposition 1 above) and the construction can be modified to show that also alphabetic SDI preserves regularity. To conclude, we consider the nondeterministic state complexity of these operations.

Lemma 4. *For NFAs M and N having, respectively, m and n states, the language $L(M) \overset{\text{a}-\text{sdi}}{\leftarrow} L(N)$ can be recognized by an NFA with $mn + 2m$ states.*

The upper bound is the same as the bound for the nondeterministic state complexity of ordinary insertion [11], however, the construction used for Lemma 4 is not the same. Using Lemma 1 (the fooling set lemma [1]) we can establish a matching lower bound.

Lemma 5. *For $m, n \in \mathbb{N}$, there exist regular languages L_1 and L_2 over a binary alphabet having NFAs with m and n states, respectively, such that any NFA for $L_1 \overset{\text{a}-\text{sdi}}{\leftarrow} L_2$ needs at least $mn + 2m$ states.*

The above lemmas establish the precise nondeterministic state complexity of alphabetic SDI.

Corollary 3. *The worst case nondeterministic state complexity of the alphabetic site-directed insertion of an n-state NFA language into an m-state NFA language is $mn+2m$. The lower bound can be reached by languages over a binary alphabet.*

It is less obvious what is the precise nondeterministic state complexity of the general SDI. If A has m states and B has n states, Proposition 1 gives an upper bound $3mn + 2m$ for the nondeterministic state complexity of $L(A) \overset{\text{sdi}}{\leftarrow} L(B)$. Likely the bound cannot be improved but we do not have a proof for the lower bound.

Problem 3. What is the nondeterministic state complexity of site-directed insertion?

References

1. Birget, J.C.: Intersection and union of regular languages and state complexity. Inform. Process. Lett. **43**, 185–190 (1992)
2. Cho, D.-J., Han, Y.-S., Ng, T., Salomaa, K.: Outfix-guided insertion. Theoret. Comput. Sci. **701**, 70–84 (2017)
3. Csuhaj-Varju, E., Petre, I., Vaszil, G.: Self-assembly of string and languages. Theoret. Comput. Sci. **374**, 74–81 (2007)
4. Daley, M., Kari, L., Gloor, G., Siromoney, R.: Circular contextual insertions/deletions with applications to biomolecular computation. In: String Processing and Information Retrieval Symposium, pp. 47–54 (1999)
5. Domaratzki, M.: Semantic shuffle on and deletion along trajectories. DLT 2004, LNCS, vol. 3340, pp. 163–174, Springer, Heidelberg (2004)
6. Domaratzki, M.: Trajectory-based codes. Acta Inf. **40**, 491–527 (2004)
7. Domaratzki, M., Rozenberg, G., Salomaa, K.: Interpreted trajectories. Fundamenta Informaticae **73**, 81–97 (2006)
8. Enaganti, S., Kari, L., Kopecki, S.: A formal language model of DNA polymerase enzymatic activity. Fundamenta Informaticae **138**, 179–192 (2015)
9. Enaganti, S., Ibarra, O., Kari, L., Kopecki, S.: On the overlap assembly of strings and languages, Natural Computing **16**, 175–185 (2017)
10. Franco, G., Manca, V.: Algorithmic applications of XPCR. Natural Computing **10**, 805–811 (2011)
11. Han, Y.-S., Ko, S.-K., Ng, T., Salomaa, K.: State complexity of insertion, Internat. J. Foundations Comput. Sci. **27**, 863-878 (2016)
12. Holzer, M., Jakobi, S.: Descriptional complexity of chop operations on unary and finite languages. J. Automata, Languages and Combinatorics **17**(2-4), 165–183 (2012)
13. Holzer, M., Jakobi, S., Kutrib, M.: The chop of languages. Theor. Comput. Sci. **682**, 122-137 (2017)
14. Jürgensen, H., Konstantinidis, S.: Codes. In: *Handbook of Formal Languages, Vol. 1.,* (Rozenberg, G., Salomaa, A., Eds.), Springer, pp. 511–607 (1997)
15. Kari, L.: On language equations with invertible operations. Theoret. Comput. Sci. **132**, 129–150 (1994)
16. Kari, L., Thierrin, G.: Contextual insertions/deletions and computability. Inform. Computation **131**, 47–61 (1996)
17. Mateescu, A., Rozenberg, G., Salomaa, A.: Shuffle on trajectories: Syntactic constraints. Theoret. Comput. Sci. **197**, 1–56 (1998)
18. Reikofski, J., Yao, B.Y.: Polymerase chain reaction (PCR) techniques for site-directed mutagenesis. Biotechnology Advances **10**, 535–547 (1992)
19. Shallit, J.: *A Second Course in Formal Languages and Automata Theory,* Cambridge University Press (2009)