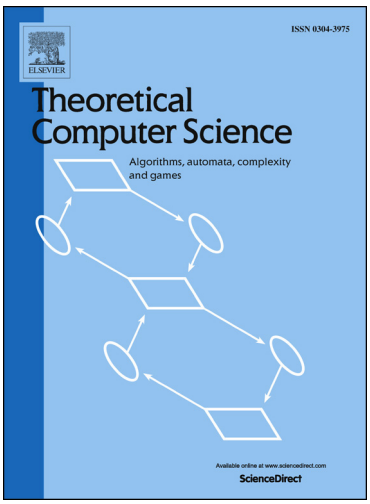
Accepted Manuscript



Site-Directed Insertion: Language Equations and Decision Problems

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

PII: S0304-3975(19)30307-X
DOI: https://doi.org/10.1016/j.tcs.2019.04.019
Reference: TCS 12005

To appear in: *Theoretical Computer Science*

Received date: 4 November 2018
Revised date: 15 April 2019
Accepted date: 20 April 2019

Please cite this article in press as: D.-J. Cho et al., Site-Directed Insertion: Language Equations and Decision Problems, *Theoret. Comput. Sci.* (2019), https://doi.org/10.1016/j.tcs.2019.04.019

This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.

ACCEPTED MANUSCRIPT

Site-Directed Insertion: Language Equations and Decision Problems

Da-Jung Cho[a,b], Yo-Sub Han[c], Kai Salomaa[d], Taylor J. Smith[d]

[a]*CNRS & LSV, ENS Paris-Saclay, 61, avenue du Président Wilson, CACHAN Cedex 94235, France*
[b]*LRI, Université Paris-Sud, Bât 650, Rue Noetzlin, Gif-sur-Yvette 91190, France*
[c]*Department of Computer Science, Yonsei University, 50, Yonsei-Ro, Seodaemun-Gu, Seoul 120-749, Republic of Korea*
[d]*School of Computing, Queen's University, Kingston, Ontario K7L 2N8, Canada*

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 (M. Domaratzki, *Developments in Language Theory* 2004) we show that one variable equations involving site-directed insertion and regular constants can be solved algorithmically. We consider also maximal and minimal variants of the site-directed insertion operation and the nondeterministic state complexity of site-directed insertion.

Keywords: Finite automata, language operations, shuffle on trajectories, decidability, state complexity

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 [25]. The algorithmic applications of mutagenesis have been considered e.g. by Franco and Manca [13]. Contextual insertion/deletion systems in the study of molecular computing have been used, e.g. by Kari and Thierrin [22], Daley et al. [6] and Enaganti et al. [11].

An outfix of a string consists of a prefix and suffix of the string that do not overlap. Site-directed insertion (SDI) of a string y into a string x involves matching an outfix of y

[☆]A preliminary version of this paper [4] was presented at the 20th International Conference Descriptional Complexity of Formal Systems held in Halifax, Canada, July 25–27, 2018. 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), and the CNRS project PEPS DEMO. Han was supported by the Basic Science Research Program (NRF-2018R1D1A1A09084107). Salomaa and Smith were supported by Natural Sciences and Engineering Research Council of Canada Grant OGP0147224.

Email addresses: da-jung.cho@lsv.fr,dajung.cho@lri.fr (Da-Jung Cho), emmous@yonsei.ac.kr (Yo-Sub Han), ksalomaa@cs.queensu.ca (Kai Salomaa), tsmith@cs.queensu.ca (Taylor J. Smith)

ACCEPTED MANUSCRIPT

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* [3]. 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 [2, 5, 12, 15, 16].

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. [16].

Site-directed insertion can be represented as a *semantic shuffle on trajectories* (SST). Shuffle on trajectories was introduced by Mateescu et al. [24] and the extension to SST is due to Domaratzki [7]. Further extensions of the shuffle-on-trajectories operation have been studied by Domaratzki et al. [9].

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 guarantees the existence of regularity preserving left- and right-inverses of the operation. By the general results of Kari [20] on the decidability of language equations, as translated for SST by Domaratzki [7], 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 [7] (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 [18] can be decided in a polynomial time. The decidability of the question whether a regular language is closed under maximal (or minimal) SDI remains open.

In the last section consider the nondeterministic state complexity of the SDI operations. 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 nondeterministic 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 [26]. 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 $q \in Q$

ACCEPTED MANUSCRIPT

and $a \in \Sigma$. It is well known that the deterministic and nondeterministic finite automata recognize the class of *regular languages.*

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 $\odot$-independence properties are 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 [8] 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 [18].

3. Site-Directed Insertion

The site-directed insertion is a partially overlapping insertion operation, analogously as the overlap-assembly (or self-assembly) [2, 5, 12] models an overlapping concatenation of strings. The overlapping concatenation operation is also called the chop operation [16].

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 string 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 [3] uses the name "outfix-guided insertion" for the same operation. The current name is better consistent with terminology used in applications to site-directed mutagenesis [13, 25].

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

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

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 [15] but the later definition of the chop-operation [16] coincides with general overlap-assembly

ACCEPTED MANUSCRIPT

[12]. 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

$$x \overset{\text{a-sdi}}{\leftarrow} y = \{x_1 azbx_2 \mid x = x_1 abx_2,\ y = azb,\ a, b \in \Sigma,\ x_1, x_2, z \in \Sigma^*\}.$$

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, whereas the context-free languages are not closed under general site-directed insertion [3].

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 fact that regular languages are closed under site-directed insertion and operates in polynomial time when L is specified by a DFA [3]. Deciding whether a context-free language is closed under site-directed insertion is undecidable [3].

A language L is $\overset{\text{sdi}}{\leftarrow}$-free, or SDI-free, with respect to a language 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 [18] 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 $L(B)$.

Proof. (i) We note that $L(A) \overset{\text{sdi}}{\leftarrow} L(B) = \emptyset$ if and only if for any nontrivial outfix (u, v) of a string of $L(B)$, the concatenation $u \cdot v$ is not a substring of a string of $L(A)$. Suppose that A has m states and B has n states. The set of strings uv, where (u, v) is a nontrivial outfix of a string of $L(B)$ can be recognized by an NFA B' that operates as follows: (ia) B' simulates B on a prefix of the input and counts that the prefix has length at least one, (ib) then B' makes a nondeterministic transition to any state that is reachable from the current state of B in the simulated computation, and, (ic) simulates an accepting computation of B on a suffix of length at least one. In total the construction uses $4 \cdot n$ states. The set of substrings of $L(A)$ has an NFA A' with $m + 1$ states. Now $L(A) \overset{\text{sdi}}{\leftarrow} L(B) = \emptyset$ if and only if $L(A') \cap L(B') = \emptyset$ and emptiness of NFAs can be decided in polynomial time.

We note that $L(A) \overset{\text{sdi}}{\leftarrow} \Sigma^+$ consists of $L(A)$ and the strings obtained from $w \in L(A)$ by inserting a nonempty string somewhere "in the middle" of w. Consequently $L(A) \overset{\text{sdi}}{\leftarrow} \Sigma^+$ has an NFA A' with $3m + 1$ states and deciding SDI independence with respect to $L(B)$ can be done by checking emptiness of $L(A') \cap L(B)$.

ACCEPTED MANUSCRIPT

(ii) Now $L(A) \overset{\text{a-sdi}}{\leftarrow} L(B) = \emptyset$ if and only if for some outfix (b, c) of $L(B)$, where b and c are elements of the alphabet Σ, $\Sigma^* bc\Sigma^* \cap L(A) \neq \emptyset$. These conditions can be checked using a simple modification of the above proof.

Finally, checking alphabetic SDI-independence is the same as in (i) because $L(A) \overset{\text{a-sdi}}{\leftarrow} \Sigma^+ = L(A) \overset{\text{sdi}}{\leftarrow} \Sigma^+$. □

It is, perhaps, not very surprising that for context-free languages deciding SDI-freeness and SDI-independence is undecidable.

Proposition 1. *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.

Proof. Recall that an instance of the *Post Correspondence Problem* (PCP) consists of two sequences of strings $(u_1, \ldots, u_m)$ and $(v_1, \ldots, v_m)$ over an alphabet Σ. A *solution* for the instance is a sequence of indices $(i_1, \ldots, i_k)$, $1 \leq i_j \leq m$, such that $u_{i_1} \cdots u_{i_k} = v_{i_1} \cdots v_{i_k}$ [26].

(i) Choose $\Sigma = \{a, b\}$. For a PCP instance as above define context-free languages $L_1, L_2 \subseteq (\Sigma \cup \{\#, \$, \%\} \cup \{1, \ldots, m\})^*$ by setting

$$L_1 = \{\#i_1 i_2 \cdots i_r \$\$ u_{i_r} u_{i_{r-1}} \cdots u_{i_1} \# \mid 1 \leq i_1, i_2, \ldots, i_r \leq m\}, \text{ and,}$$

$$L_2 = \{\#j_1 j_2 \cdots j_s \$\%\$ v_{j_s} v_{j_{s-1}} \cdots v_{j_1} \# \mid 1 \leq j_1, j_2, \ldots, j_s \leq m\}.$$

We claim that L_1 is SDI-free with respect to L_2 if and only if the PCP instance does not have a solution. Clearly, if $w_1 \in L_1$ and $w_2 \in L_2$ are strings encoding the same solution for the PCP instance, then w_2 can be inserted into w_1 (using the entire string w_1 as the insertion guide) and $w_1 \overset{\text{sdi}}{\leftarrow} w_2 \neq \emptyset$.

For the converse implication, consider a site-directed insertion of

$$w_2 = \#j_1 j_2 \cdots j_s \$\%\$ v_{j_s} v_{j_{s-1}} \cdots v_{j_1} \# \in L_2$$

into $w_1 = \#i_1 i_2 \cdots i_r \$\$ u_{i_r} u_{i_{r-1}} \cdots u_{i_1} \# \in L_1$. Since the strings w_1 and w_2 have occurrences of # only as the first and last symbol, the entire string w_1 must be matched with an outfix of w_2. Since w_1 has two occurrences of the symbol \$, the only possibility for the matched outfix is $(\#j_1 \cdots j_s \$, \$ v_{j_s} v_{j_{s-1}} \cdots v_{j_1} \#)$. Thus, $j_1 j_2 \cdots j_s = i_1 i_2 \cdots i_r$ and $v_{j_s} \cdots v_{j_1} = u_{i_r} \cdots u_{i_1}$ and and the strings w_1 and w_2 encode a solution $(j_s, j_{s-1}, \ldots, j_1)$ of the PCP instance.[1]

(ii) Let L_1 and L_2 be the context-free languages defined above in (i), denote their alphabet by Ω and let $\Omega' = \Omega - \{\#, \$, \%\}$. We note that $L_1 \subseteq \#\Omega'^* \$\$ \Omega'^* \#$ and $L_2 \subseteq \#\Omega'^* \$\% \$ \Omega'^* \#$. Thus, to obtain a string $w_2 \in L_2$ by inserting a string in Ω^+ to a string $w_1 \in L_1$ the only possibility is that the insertion has to add the single occurrence of %, and the prefixes

[1] In the reversal of L_i the sequence of indices would directly encode the PCP solution but the reversal of L_i is not deterministic context-free.

ACCEPTED MANUSCRIPT

(respectively, suffixes) of w_1 and w_2 in $\#\Omega'^*\$$ (respectively, in $\$\Omega'^*\#$) have to match. This is only possible if the PCP instance has a solution. Thus, L_1 is SDI-independent with respect to L_2 if and only if the PCP instance does not have a solution. □

The languages L_1 and L_2 used in the proof of Proposition 1 are, in fact, deterministic context-free and the proof implies that SDI-freeness and SDI-independence are undecidable for deterministic context-free languages.

On the other hand, the positive decidability result of Theorem 1 can be extended to the case where one of the languages is context-free and the other is regular. For context-free L and regular R, the languages $L \overset{\text{sdi}}{\leftarrow} R$ and $R \overset{\text{sdi}}{\leftarrow} L$ are context-free and the size of the pushdown automaton obtained from the construction of Theorem 5.3 of [3] is polynomial in the size of the pushdown automaton and NFA specifying L and R, respectively. Then deciding whether L is SDI-free with respect to R (or R is SDI-free with respect to L) can be done using the polynomial time algorithm for context-free language emptiness [17].

Similarly, to decide SDI-independence of L with respect to R (or vice versa) we can construct a pushdown automaton for $(L \overset{\text{sdi}}{\leftarrow} \Sigma^+) \cap R$ (respectively, for $(R \overset{\text{sdi}}{\leftarrow} \Sigma^+) \cap L$) and decide emptiness in polynomial time.

3.2. Language equations

For dealing with language equations we express the site-directed insertion operation as a *semantic shuffle on a set of trajectories,* SST (Domaratzki [7]). The semantic shuffle extends the (syntactic) *shuffle on trajectories* originally defined by Mateescu et al. [24]. We use a simplified definition of SST that does not allow *content restriction* [7].

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

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

ACCEPTED MANUSCRIPT

Intuitively, the trajectory t is a sequence of instructions that guide the shuffling of the argument 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 [24]) and σ represents synchronized insertion where the next symbols of the argument strings must coincide.

For $x, y \in \Sigma^*$ and $t \in \Gamma^*$, $x \Cap_t y$ either consists of a unique string or is undefined. For $T \subseteq \Gamma^*$, $x \Cap_T y = \bigcup_{t \in T} x \Cap_t y$ and the operation is extended in the natural way for languages over Σ. Note that $x \Cap_{\{0,1\}^*} y$ is the unrestricted shuffle of the strings x and y.

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 2. *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,

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

Now using the strong decidability results of Domaratzki [7] 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 [20] 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* [7]. 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.

Definition 1. (Domaratzki [7]) Let $\Delta = \{i, d, \sigma\}$. The *deletion* of a string $y \in \Sigma^*$ from $x \in \Sigma^*$ *along a trajectory* $t \in \Delta^*$, denoted $x \rightsquigarrow_t y$, is defined as follows.

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

$$x \rightsquigarrow_t \varepsilon = \begin{cases} a(x' \rightsquigarrow_{t'} \varepsilon) \text{ if } c = i, \\ \emptyset, \text{ otherwise.} \end{cases}$$

Finally, for the general case $x = ax'$, $y = by'$, $a, b \in \Sigma$, $t = ct'$, $c \in \Delta$,

$$x \rightsquigarrow_t y = \begin{cases} a(x' \rightsquigarrow_{t'} y) \text{ if } c = i, \\ x' \rightsquigarrow_{t'} y' \text{ if } c = d \text{ and } a = b, \\ a(x' \rightsquigarrow_{t'} y') \text{ if } c = \sigma \text{ and } a = b, \\ \emptyset, \text{ otherwise.} \end{cases}$$

For $x, y \in \Sigma^*$ and $T \subseteq \Delta^*$, $x \rightsquigarrow_T y = \bigcup_{t \in T} x \rightsquigarrow_t y$.

ACCEPTED MANUSCRIPT

The deletion along trajectories operation is again extended in the natural way for sets of trajectories and for languages. The set $x \rightsquigarrow_{i^*d^*i^*} y$ consists of strings obtained from x by deleting a substring y and $x \rightsquigarrow_{\{i,d\}^*} y$ gives the scattered deletion of y from x [21].

We can express the left- and right-inverse (as defined by Kari [20], see also [7]) 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 $\diamond^{\mathrm{rev}}$ be the operation defined by $x \diamond^{\mathrm{rev}} y = y \diamond x$ for all $x, y \in \Sigma^*$.

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) $X \overset{\mathrm{sdi}}{\leftarrow} L = R$, (b) $L \overset{\mathrm{sdi}}{\leftarrow} X = R$, (c) $X \overset{\mathrm{a-sdi}}{\leftarrow} L = R$, (d) $L \overset{\mathrm{a-sdi}}{\leftarrow} X = R$.*

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) $S_b = \overline{L(\rightsquigarrow_{T_2})^{\mathrm{rev}}\overline{R}}$, (c) $S_c = \overline{\overline{R} \rightsquigarrow_{T_1^a} L}$, (d) $S_d = \overline{L(\rightsquigarrow_{T_2^a})^{\mathrm{rev}}\overline{R}}$.

Proof. By Proposition 2, SDI is represented by semantic shuffle on $T_{\mathrm{sdi}} = 0^*\sigma^+1^*\sigma^+0^*$ and alphabetic SDI is represented by semantic shuffle on $T_{\mathrm{a-sdi}} = 0^*\sigma1^*\sigma0^*$.

For (a) we observe (as in Theorem 6.1 of [7]) that the operations $\rightsquigarrow_{T_1}$ and $\pitchfork_{T_{\mathrm{sdi}}}$ are *left-inverses*[2] of each other in the sense that for all strings x, y, z,

$$x \in y \pitchfork_{T_{\mathrm{sdi}}} z \quad \text{iff} \quad y \in x \rightsquigarrow_{T_1} z.$$

Thus by Theorem 6.4 of Domaratzki [7] (or Thms. 4.2 and 4.6 of Kari [20]), if $X \pitchfork_{T_{\mathrm{sdi}}} L = R$ has a solution, S_a is the unique maximal solution. By Proposition 2 this equation is the same as $X \overset{\mathrm{sdi}}{\leftarrow} L = R$. Since semantic shuffle and deletion along trajectories preserve regularity, the equality $S_a \overset{\mathrm{sdi}}{\leftarrow} L = R$ can be effectively tested.

Similarly for (b), we observe that the operations $\pitchfork_{T_{\mathrm{sdi}}}$ and $(\rightsquigarrow_{T_2})^{\mathrm{rev}}$ are *right-inverses* in the sense that for all strings x, y, z,

$$x \in y \pitchfork_{T_{\mathrm{sdi}}} z \quad \text{iff} \quad z \in x \rightsquigarrow_{T_2} y \quad \text{iff} \quad z \in y(\rightsquigarrow_{T_2})^{\mathrm{rev}} x.$$

The first "iff" is verified as in Theorem 6.2 of [7] and the second "iff" just inverts the operation.

Again by the same results of [7] (based on [20]), if $L \pitchfork_{T_{\mathrm{sdi}}} X = R$ (i.e., $L \overset{\mathrm{sdi}}{\leftarrow} X = R$) has a solution, S_b is the unique maximal solution.[3] Since semantic shuffle and deletion along trajectories preserve regularity, the equality $L \pitchfork_{T_{\mathrm{sdi}}} S_b = R$ can be effectively tested.

The case (c) follows similarly from the observation that the relations $\overset{\mathrm{a-sdi}}{\leftarrow}$ $(=\pitchfork_{T_{\mathrm{a-sdi}}})$ and $\rightsquigarrow_{T_1^a}$ are left-inverses of each other, and (d) from the observation that $\overset{\mathrm{a-sdi}}{\leftarrow}$ $(=\pitchfork_{T_{\mathrm{a-sdi}}})$ and $(\rightsquigarrow_{T_2^a})^{\mathrm{rev}}$ are right-inverses of each other. □

[2]Note that the left-inverse and right-inverse relations are symmetric.

[3]Naturally, the language S_b could be expressed using $\rightsquigarrow_{T_2}$, instead of $(\rightsquigarrow_{T_2})^{\mathrm{rev}}$. We use the latter notation (although it looks more complicated), because $(\rightsquigarrow_{T_2})^{\mathrm{rev}}$ is the right-inverse of $\overset{\mathrm{sdi}}{\leftarrow}$ and this makes the connection with [7, 20] more transparent.

ACCEPTED MANUSCRIPT

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 [24].

It seems unlikely that the decidability results of Theorem 2 could be extended to the case where one of the constant languages is regular and the other is context-free. If $\leftarrow$ denotes the usual (unrestricted) insertion operation, it is known that, for context-free L and regular R, the existence of solutions for an equation $X \leftarrow L = R$ or $L \leftarrow X = R$ is undecidable [19, 23]. Similar undecidability results hold for deletion, shuffle and other standard operations [19, 23]. We leave it as an exercise for the reader to determine if the undecidability results can be extended for site-directed insertion.

The known trajectory based methods for two variable equations [7] 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. [16] define two deterministic variants of the chop operation (a.k.a. overlap assembly [2, 12]). 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.

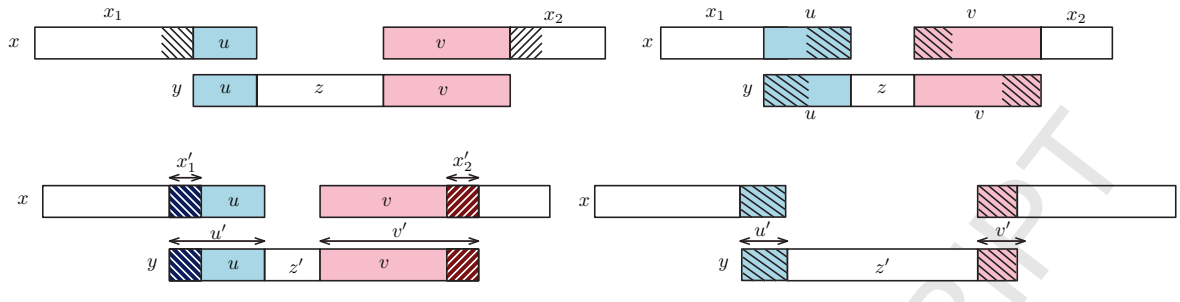
Formally, the *maximal site-directed insertion* (max-SDI) of a string y into a string x is defined as follows:

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

Equivalently the maximal SDI of x and y is

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

ACCEPTED MANUSCRIPT



(a) Definition of maximal SDI

(b) Definition of minimal SDI

Figure 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:

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

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.

ACCEPTED MANUSCRIPT

Lemma 1. *Let $x, y \in \Sigma^*$.*

(i) $x \overset{\max-\text{sdi}}{\leftarrow} y \subseteq x \overset{\text{sdi}}{\leftarrow} y$ *and* $x \overset{\text{a}-\text{sdi}}{\leftarrow} y \subseteq x \overset{\min-\text{sdi}}{\leftarrow} y \subseteq x \overset{\text{sdi}}{\leftarrow} y$.

(ii) $x \overset{\text{sdi}}{\leftarrow} y \neq \emptyset$ *iff* $x \overset{\max-\text{sdi}}{\leftarrow} y \neq \emptyset$ *iff* $x \overset{\min-\text{sdi}}{\leftarrow} y \neq \emptyset$.

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

Proof. (i) The minimal SDI and the maximal SDI operations are special cases of site-directed insertion. An alphabetic site-directed insertion of y into x is necessarily minimal because the outfix-guide of the operation must be non-trivial.

For (ii) we note that if $x \overset{\text{sdi}}{\leftarrow} y \neq \emptyset$ where the insertion is guided by an outfix (u, v), and the insertion is not maximal, then we can extend either u or v to obtain a "larger" outfix guide. Since y is a finite string the process must terminate. The "only if" holds because maximal SDI is a special case of general SDI. Finally, the claim for minimal SDI is analogous.

For (iii) we note that $aba \in aba \overset{\min-\text{sdi}}{\leftarrow} aba$ but $aba \overset{\text{a}-\text{sdi}}{\leftarrow} aba = \emptyset$.

Since the max-chop and min-chop operations do not preserve regularity [16], 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 [16].

Proposition 3. *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

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

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 $x \overset{\max-\text{sdi}}{\leftarrow} y$ is in L_{result}.

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

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

For non-closure under min-SDI we claim that

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

ACCEPTED MANUSCRIPT

Consider $x = ba^iba^j\$ \in L_1$ and $y = ba^rba^s\%\$ \in L_2$. In order for the result of site-directed insertion of y into x to have two b's, ba^iba^j must be a prefix of ba^rba^s, that is, $i = r$ and $j \leq s$. For the site-directed insertion to be minimal, no proper non-empty prefix of ba^iba^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 $n > m$, $ba^mba^n\%\$ \in ba^mba^n\$ \overset{\text{min}-\text{sdi}}{\leftarrow} ba^mba^n\%\$$. □

In fact, extending the max-chop and min-chop constructions from Theorem 3 of [16] it would be possible to show that there exist regular languages L_1 and L_2 such that $L_1 \overset{\text{max}-\text{sdi}}{\leftarrow} L_2$ (or $L_1 \overset{\text{min}-\text{sdi}}{\leftarrow} L_2$) 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 4. *Let R be a regular language and L a finite language. Then the languages $R \overset{\text{max}-\text{sdi}}{\leftarrow} L$, $R \overset{\text{min}-\text{sdi}}{\leftarrow} L$, $L \overset{\text{max}-\text{sdi}}{\leftarrow} R$, and $L \overset{\text{min}-\text{sdi}}{\leftarrow} R$ are regular.*

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

Since

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

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 $L(A) \overset{\text{max}-\text{sdi}}{\leftarrow} y$. On an input w, B nondeterministically guesses a decomposition $w = x_1y_1y_2y_3x_2$ where $x_1y_1y_3x_2 \in L(A)$, $y_1y_2y_3 = y$ and $y_1, y_3 \neq \varepsilon$. When reading the prefix x_1y_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_3x_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'_1y_1 is a prefix of y_1y_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, $y_3x'_2$ is a suffix of y_2y_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 $x_1y_1y_2y_3x_2$ gives a maximal site-directed insertion of y into a string of $L(A)$. □

4.1. Decision problems for maximal/minimal SDI

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

ACCEPTED MANUSCRIPT

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 1 (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 general site-directed insertion preserves regularity [3], for regular languages L_1 and L_2 the membership problem for $L_1 \overset{\text{sdi}}{\leftarrow} L_2$ can be decided in polynomial time. From Proposition 3 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 still 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 2. *For NFAs A and B and $w \in \Sigma^*$ we can decide in polynomial time whether $w \in L(A) \overset{\text{min-sdi}}{\leftarrow} L(B)$, or whether $w \in L(A) \overset{\text{max-sdi}}{\leftarrow} L(B)$.*

Proof. We describe an algorithm that decides membership in $L(A) \overset{\text{min-sdi}}{\leftarrow} L(B)$ (the algorithm for maximal SDI is similar).

The string w has $O(|w|^4)$ decompositions into five parts $w = x_1uzvx_2$, $u, v \neq \varepsilon$. For each decomposition the algorithm verifies that $x_1uvx_2 \in L(A)$, $uzv \in L(B)$ and that the strings u and v are unbordered. Based on the Morris-Pratt pattern matching algorithm checking that a string is unbordered can be done in linear time [10]. The uniform membership problem for NFAs is in polynomial time. □

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 4 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.*

ACCEPTED MANUSCRIPT

Proof. By Proposition 4 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 4 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 4 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 a regular set of trajectories. Thus, the tools [7] that were used in section 3.2 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 deterministic state complexity of insertion and of overlap assembly has been determined, respectively, in [14] and [2]. Site-directed insertion can be viewed as a non-trivial generalization of these operations and obtaining good upper and lower bounds for the deterministic state complexity of SDI remains a challenging problem.

In this section, we consider the nondeterministic state complexity of SDI and alphabetic SDI. The construction of Lemma 4.1 of [3] gives an upper bound for the nondeterministic state complexity of SDI.

Proposition 5. *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.*

We conjecture that, for NFAs A and B with, respectively, m and n states, the upper bound $3mn + 2m$ for the size of an NFA for $L(A) \overset{\text{sdi}}{\leftarrow} L(B)$ cannot be improved. However, we do not a have proof for a lower bound.

Next we give a tight bound for the nondeterministic state complexity of alphabetic SDI. The bound is the same as the bound for the nondeterministic state complexity of ordinary insertion [14], however, neither the upper bound construction of Lemma 3 nor the lower bound argument in Lemma 5 is the same.

Lemma 3. *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.*

ACCEPTED MANUSCRIPT

Proof. Let $M = (Q_M, \Sigma, \delta, s_M, F_M)$ and $N = (Q_N, \Sigma, \gamma, s_N, F_N)$ be NFAs recognizing L_1 and L_2 with m and n states. Given NFAs M and N, we construct a new NFA $A = (Q_A, \Sigma, \Omega, s_A, F_A)$, where

$$Q_A = Q_M \cup (Q_M \cup \{\clubsuit\} \times Q_N) \cup \widehat{Q_M},$$

$s_A = s_M$, $\widehat{q_f} \in F_A$. We notice that the set $\widehat{Q_M}$ of states is a copy of the set Q_M of states of A. Transitions of Ω are defined as:

(i) for $q \in Q_M$ and $\alpha \in \Sigma$:

$$\Omega(q, \alpha) = \{q' \mid q' \in Q_M \text{ and } q' \in \delta(q, \alpha)\} \cup \{(q', s_M) \mid q' \in \delta(q, \alpha)\}$$

(ii) for (q, s_N), where $q \in Q_M$:

$$\Omega((q, s_N), \alpha) = \{(q_t, p) \mid q_t \in \delta(q, \alpha) \text{ and } p \in \gamma(s_M, \alpha)\} \cup \{(\clubsuit, p') \mid p \in \gamma(s_N, \alpha)\}$$

(iii) for $(\clubsuit, p)$, where $p \in Q_N$:

$$\Omega((\clubsuit, p), \alpha)) = \{(\clubsuit, p') \mid p' \in \gamma(p, \alpha)\} \cup \{(q_t, p') \mid p' \in \gamma(p, \alpha)\}$$

(iv) for (q_t, p), where $q_t \in Q_M$ and $p \in Q_N$:

$$\Omega((q_t, p), \alpha) = Z_A,$$

where

$$Z_A = \begin{cases} \{\widehat{q_t'} \mid q_t' \in \delta(q_t, \alpha)\} & \text{if } \gamma(p, \alpha) \subseteq F_N \\ \emptyset & \text{otherwise} \end{cases}$$

(v) for $\widehat{q} \in \widehat{Q_M}$ and $\alpha \in \Sigma$:

$$\Omega(\widehat{q}, \alpha) = \{\widehat{q'} \mid q' \in \delta(q, \alpha)\}$$

We consider a string $w = x_1 azbx_2 \in L(M) \overset{a-sdi}{\leftarrow} L(N)$, where $x_1 abx_2 \in L(M), azb \in L(N)$, $z \in \Sigma^*$ and $a, b \in \Sigma$. NFA A uses the set Q_M of states for simulating a computation $comp_M(x_1)$ of M, the set $(Q_M \times Q_N)$ of states for simulating two computations $comp_M(a)$ and $comp_N(b)$, the set $(\clubsuit \times Q_N)$ of states for simulating a computation $comp_N(z)$, and the set $\widehat{Q_M}$ of states for simulating a computation $comp_M(x_2)$ (See Figure 2.)

The simulation begins in the initial state s_M. The transitions (i) simulate a computation $comp_M(x_1)$ of M. After A reaches a state $q \in Q_M$, the transitions (i) can make a λ-transition to state (q, s_M), which means that A nondeterministically guesses a state $q \in Q_M$ to simulate computations $comp_M(a)$ and $comp_N(a)$. When A reaches a state (q, s_M), the transitions (ii) simulate a computation of both M and N from q to q_t and from s_M to p, respectively. Note that the alphabetic site-directed insertion of $L(N)$ into $L(M)$ has an insertion guide (a, b) of length 1—$a, b \in \Sigma$—if $L(M) \overset{a-sdi}{\leftarrow} L(N) \neq \emptyset$. It leads the transitions (ii)

ACCEPTED MANUSCRIPT

to make a λ-transition to state $(\clubsuit, p')$, where $p' \in \gamma(s_M, \alpha)$ after reading a character a. The transitions (iii) allows A to simulate only N from p to p' where $p \in \gamma(s_N, \alpha)$, and it can make a λ-transition to state (q_t, p') after reading z on $comp_N(azb)$, where $(q_t, p) \in \Omega((q, s_M), \alpha)$. Then, A begins to read a character b on both accepting computations $comp_M(x_1abx_2)$ and $comp_N(azb)$ following the transitions (iv). If A finishes to simulate an accepting computation $comp_N(azb)$ of N—$\gamma(p, \alpha) \subseteq F_N$—, A moves to a state $\widehat{q'_t}$. The transitions (v) simulate a remaining computation $comp_M(x_2)$ of M, which ends on state $\widehat{q}$, where $q \in F_M$. The simulation shows that A accepts a string x_1azbx_2 if M accepts a string x_1abx_2 and N accepts a string azb, where $x_1, x_2, z \in \Sigma^*$ and $a, b \in \Sigma$.

We now show that A only accepts strings of $L(M) \overset{a-sdi}{\leftarrow} L(N)$. The transitions of A guarantee that there are five parts P_1, P_2, P_3, P_4 and P_5 of an accepting computation $comp_A(x_1azbx_2)$, where P_1 uses states of Q_M, P_2 uses states of $Q_M \times Q_N$, P_3 uses states of $\{\clubsuit\} \times Q_N$, P_4 uses states of $Q_M \times Q_N$, and P_5 uses states of $\widehat{Q_M}$. States of P_1 begin to simulate only a computation of M, and the computation nondeterministically moves to states (q, s_M), where $q \in Q_M$ and $s_M \in Q_N$, for simulating a computation of both M and N. According to the transitions (ii), P_2 uses states of $(q, s_M) \subseteq Q_M \times Q_N$ to read a character $a \in \Sigma$ of $comp_A(x_1azbx_2)$, reaching a state (q', p'), where $q' \in \delta(q, \alpha)$ and $p' \in \gamma(s_M, \alpha)$. After A simulates a character of a computation of both M and N, P_3 uses states of $\{\clubsuit\} \times Q_N$ for simulating a computation of N. Note that the part P_3 may be empty if the computation goes directly to P_4 from P_2. The part P_4 begins simulation of M and N from a state $(q_t, p') \in Q_M \times Q_N$, where $(q_t, p) \in \Omega((q, s_N), a)$. The part P_5 uses states of $\widehat{Q_M}$ to simulate a computation of M, and it may be empty if the computation P_4 ends in an accepting state of the form $\widehat{q}$, where $q \in F_M$. □

Below we give a matching lower bound. For the proof we recall the fooling set technique for establishing lower bounds for nondeterministic state complexity [1]. A set P with properties as given in the statement of Lemma 4 is called a *fooling set* for the language L.

Lemma 4 (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 NFA for L has at least n states.*

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 $L_1 \overset{\text{a-sdi}}{\leftarrow} L_2$ needs at least $mn + 2m$ states.*

Proof. Let $L_1 = (a^m)^*$ and $L_2 = ((ab)^k)^* a$, where $n = 2k$. We have a fooling set $P = P_1 \cup P_2$ for $L_1 \overset{a-sdi}{\leftarrow} L_2$, where

(i) $P_1 = \{(a^i(ab)^r, (ab)^{2k-r}a^{2m-i-1}) \mid 1 \leq i \leq m \text{ and } 1 \leq r \leq 2k \leq n\}$,

(ii) $P_2 = \{(a^i(ab)^r a, b(ab)^{2k-r-1}a^{2m-i-1}) \mid 1 \leq i \leq m \text{ and } 1 \leq r \leq 2k\}$.

ACCEPTED MANUSCRIPT

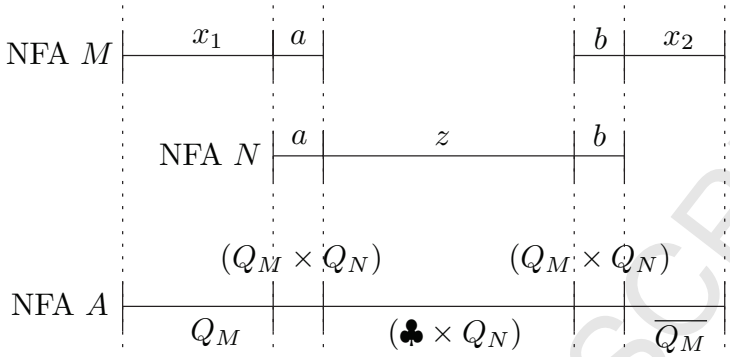


Figure 2: Let $M = (Q_M, \Sigma, \delta, s_M, F_M)$ and $N = (Q_N, \Sigma, \gamma, s_N, F_N)$ be NFAs recognizing $L_1 = \{w_1 \mid w_1 = x_1abx_2, a, b \in \Sigma \text{ and } x_1, x_2 \in \Sigma^*\}$ and $L_2 = \{w_2 \mid w_2 = azb, a, b \in \Sigma \text{ and } z \in \Sigma^*\}$, respectively. A new NFA $A = (Q_A, \Sigma, \Omega, s_M, F_A)$ recognizes $L(L_1 \overset{a-sdi}{\leftarrow} L_2)$. The sets $Q_M, \widehat{Q_M} \subseteq Q_A$ of states are used for simulating computations x_1 and x_2 of M. The set $Q_M \times Q_N \subseteq Q_A$ of states is used for simulating computations a and b of both M and N. The set $\{\clubsuit\} \times Q_N \subseteq Q_A$ of states is used for simulating a computation z of N.

Then, the resulting language can be defined by

$$L_1 \overset{a-sdi}{\leftarrow} L_2 = \{a^s((ab)^{2k})^+a^t \mid s{+}t{+}1 \equiv 0 \pmod m\}.$$

We consider two pairs

$$(x, y) = (a^i(ab)^r, (ab)^{2k-r}a^{2m-i-1}) \text{ and } (x'y') = (a^{i'}(ab)^{r'}a, b(ab)^{2k'-r'-1}a^{2m-i'-1}),$$

where $(i, k, r) \neq (i', k', r')$. We show that for any two distinct pairs $(x, y), (x', y') \in P$, at least one of the pairs (x, y') or (x', y) does not belong in $L_1 \overset{a-sdi}{\leftarrow} L_2$. We easily verify that strings $x \cdot y = a^i(ab)^r \cdot (ab)^{2k-r}a^{2m-i-1}$ and $x' \cdot y' = a^{i'}(ab)^{r'}a \cdot b(ab)^{2k'-r'-1}a^{2m-i'-1}$ are both in $L_1 \overset{a-sdi}{\leftarrow} L_2$.

On the other hand, we now consider two pairs (x, y') and (x', y). When L_2 is inserted into L_1, a non-empty outfix (a, a) of $(ab)^{2k}a$ must be matched to a substring aa of $a^{2m} \in L_1$ such that

$$a^{i-1}\underline{aa}a^{2m-i-2} \overset{[\underline{a}b(ab)^{2k-1}\underline{a}]}{\Longrightarrow} a^{i-1}ab(ab)^{2k-1}aa^{2m-i-1}.$$

We say that the resulting string must have an order of three parts $a^i, (ab)^{2k}a$, and a^{2m-i-1}, where $i{+}2m{-}i{-}1{+}1 \equiv 0 \pmod m$. Then, we verify that the string $x \cdot y' = a^i(ab)^r \cdot b(ab)^{2k'-r'-1}a^{2m-i'-1}$ does not belong in $L_1 \overset{a-sdi}{\leftarrow} L_2$, since it has an incorrect order of parts $a^i, (ab)^r, b, (ab)^{2k'-r'-1}$, where $a^{2m-i'-1}$, and $i{+}2m{-}i'{-}1{+}1 \not\equiv 0 \pmod m$. Similarly, the string $x' \cdot y = a^{i'}(ab)^{r'}a \cdot (ab)^{2k-r}a^{2m-i-1}$ is not in $L_1 \overset{a-sdi}{\leftarrow} L_2$ since it has an incorrect order $a^{i'}, (ab)^{r'}, a, (ab)^{2k-r}$, and a^{2m-i-1}, where $i'{+}2m{-}i{-}1{+}1 \equiv 0 \pmod m$. By Lemma 4 we verify that any NFA recognizing $L_1 \overset{a-sdi}{\leftarrow} L_2$ has at least $mn + 2m$ states. □

ACCEPTED MANUSCRIPT

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.*

6. Conclusion

We have shown that one variable language equations involving site-directed insertion and regular constants can be solved effectively. The main open problem remaining is whether language equations involving maximal (or minimal) site-directed insertion are solvable.

Also the deterministic state complexity of site-directed insertion remains open. Site-directed insertion can be viewed as a combination of ordinary insertion and the overlap assembly operations. The state complexity of insertion was determined by Han et al. [14] and the state complexity of overlap assembly was recently determined by Brzozowski et al. [2].

[1] J.C. Birget, Intersection and union of regular languages and state complexity, Inform. Process. Lett. 43 (1992) 185–190.

[2] J. Brzozowski, L. Kari, B. Li, M. Szykula, State complexity of overlap assembly, in *Implementation and Application of Automata,* CIAA 2018, Lect. Notes Comput. Sci. 10977, Springer, 2018, pp. 109–120.

[3] D.-J. Cho, Y.-S. Han, T. Ng, K. Salomaa, Outfix-guided insertion, Theoret. Comput. Sci. 701 (2017) 70–84.

[4] D.-J. Cho, Y.-S. Han, T. K. Salomaa, T. Smith, Site-directed insertion: Decision problems, maximality and minimality, in DCFS 2018 (S. Konstantinidis, G. Pighizzini, Eds.), Lect. Notes Comput. Sci. 10192, Springer 2018, pp. 49–61.

[5] E. Csuhaj-Varju, I. Petre, G. Vaszil, Self-assembly of string and languages, Theoret. Comput. Sci. 374 (2007) 74–81.

[6] M. Daley, L. Kari, G. Gloor, R. Siromoney, Circular contextual insertions/deletions with applications to biomolecular computation, in: String Processing and Information Retrieval Symposium, 1999, pp. 47–54.

[7] M. Domaratzki, Semantic shuffle on and deletion along trajectories, in DLT 2004, Lect. Notes Comput. Sci. 3340, Springer, Heidelberg 2004, pp. 163–174.

[8] M. Domaratzki, Trajectory-based codes, Acta Inf. 40 (2004) 491–527.

[9] M. Domaratzki, G. Rozenberg, K. Salomaa, Interpreted trajectories, Fundamenta Informaticae 73 (2006) 81–97.

[10] J.-P. Duval, T. Lecroq, A. Lefebvre, Linear computation of unbordered conjugate on unordered alphabet, Theoret. Comput. Sci. 522 (2014) 77–84.

[11] S. Enaganti, L. Kari, S. Kopecki, A formal language model of DNA polymerase enzymatic activity, Fundamenta Informaticae 138 (2015) 179–192.

[12] S. Enaganti, O. Ibarra, L. Kari, S. Kopecki, On the overlap assembly of strings and languages, Natural Computing 16 (2017) 175–185.

[13] G. Franco, G., V. Manca, Algorithmic applications of XPCR, Natural Computing 10 (2011) 805–811.

[14] Y.-S. Han, S.-K. Ko, T. Ng, K. Salomaa, State complexity of insertion, Internat. J. Foundations Comput. Sci. 27 (2016) 863-878.

[15] M. Holzer, S. Jakobi, Descriptional complexity of chop operations on unary and finite languages, J. Automata, Languages and Combinatorics 17(2-4) (2012) 165–183.

[16] M. Holzer, S. Jakobi, S., M. Kutrib, The chop of languages, Theoret. Comput. Sci. 682 (2017) 122-137.

ACCEPTED MANUSCRIPT

[17] J.E. Hopcroft, J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation,* Addison-Wesley, 1979.
[18] H. Jürgensen, S. Konstantinidis, Codes, in: *Handbook of Formal Languages, Vol. 1.,* (G. Rozenberg, A. Salomaa, Eds.), Springer, 1997, pp. 511–607.
[19] L. Kari, On insertion and deletion in formal languages, Ph.D. thesis, University of Turku, 1991.
[20] L. Kari, On language equations with invertible operations, Theoret. Comput. Sci. 132 (1994) 129–150.
[21] L. Kari, Power of controlled insertion and deletion, in *Results and Trends in Theoretical Computer Science,* Lect. Notes Comput. Sci. 812, Springer, 1994, pp. 197–212.
[22] L. Kari, G. Thierrin, Contextual insertions/deletions and computability, Inform. Computation 131 (1996) 47–61.
[23] L. Kari, P. Sosík, Aspects of shuffle and deletion on trajectories, Theoret. Comput. Sci. 332 (2005) 47–61.
[24] A. Mateescu, G. Rozenberg, A. Salomaa, Shuffle on trajectories: Syntactic constraints, Theoret. Comput. Sci. 197 (1998) 1–56.
[25] J. Reikofski, B.Y. Yao, Polymerase chain reaction (PCR) techniques for site-directed mutagenesis, Biotechnology Advances 10 (1992) 535–547.
[26] J. Shallit, *A Second Course in Formal Languages and Automata Theory,* Cambridge University Press, 2009.

ACCEPTED MANUSCRIPT