Outfix-Guided Insertion
(Extended Abstract)

Da-Jung Cho[1], Yo-Sub Han[1], Timothy Ng[2], and Kai Salomaa[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, ON K7L 2N8, Canada
{ng,ksalomaa}@cs.queensu.ca

Abstract. Motivated by work on bio-operations on DNA strings, we consider an outfix-guided insertion operation that can be viewed as a generalization of the overlap assembly operation on strings studied previously. As the main result we construct a finite language L such that the outfix-guided insertion closure of L is nonregular. We consider also the closure properties of regular and (deterministic) context-free languages under the outfix-guided insertion operation and decision problems related to outfix-guided insertion. Deciding whether a language recognized by a deterministic finite automaton is closed under outfix-guided insertion can be done in polynomial time.

Keywords: Language operations · Closure properties · Regular languages

1 Introduction

Gene insertion and deletion are basic operations occurring in DNA recombination in molecular biology. Recombination creates a new DNA strand by cutting, substituting, inserting, deleting or combining other strands. Possible errors in this process directly affect DNA strands and impair the function of genes. Errors in DNA recombination cause mutation that plays a part in normal and abnormal biological processes such as cancer, the immune system, protein synthesis and evolution [1]. Since mutational damage may or may not be easily identifiable, researchers deliberately generate mutations so that the structure and biological activity of genes can be examined in detail. *Site-directed mutagenesis* is one of the most important techniques in laboratory for generating mutations on specific sites of DNA using PCR (polymerase chain reaction) based methods [7,11]. For a site-directed insertion mutagenesis by PCR, the mutagenic primers are typically designed to include the desired change, which could be base addition [15,16]. This enzymatic reaction occurs in the test tube with a DNA strand and predesigned primers in which the DNA strand includes a target region, and a predesigned

S. Brlek and C. Reutenauer (Eds.): DLT 2016, LNCS 9840, pp. 102–113, 2016.
DOI: 10.1007/978-3-662-53132-7_9

primer includes a complementary region of the target region. The complementary region of primers leads it to hybridize the target DNA region and generate a desired insertion on a specific site as a mutation. Figure 1 illustrates the procedure of site-directed insertion mutagenesis by PCR.

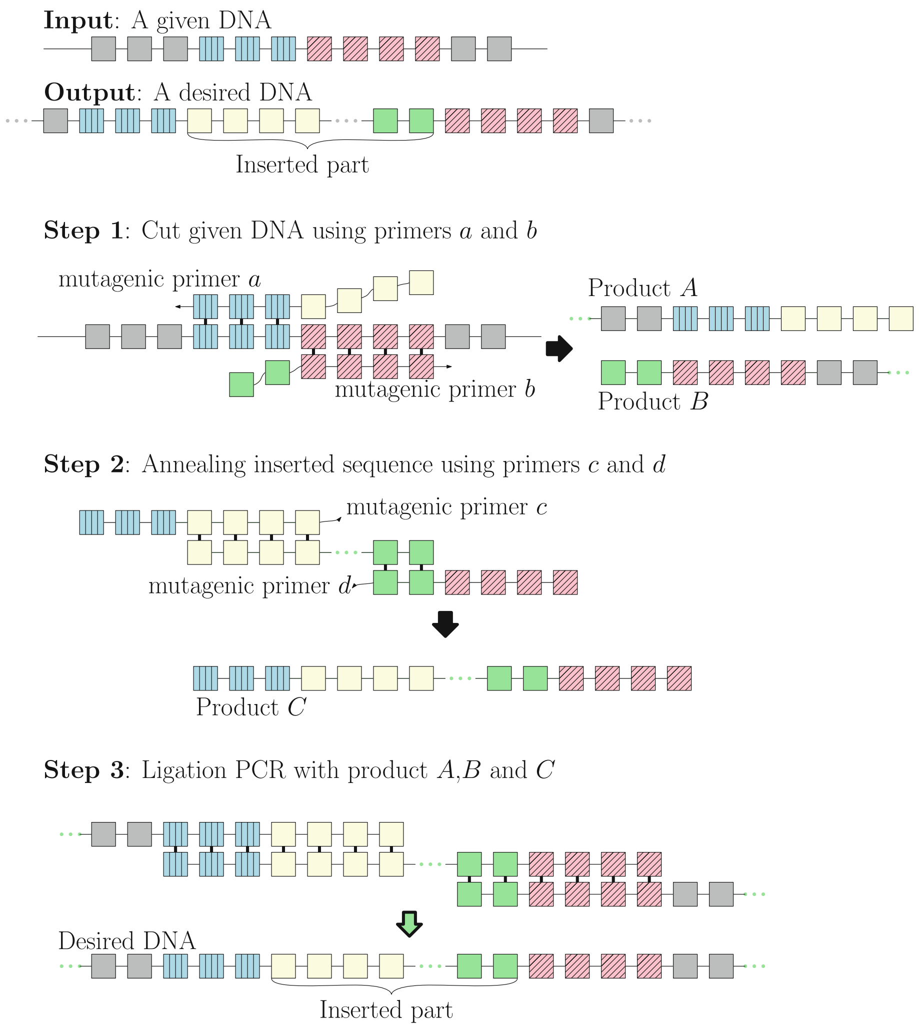
Input: A given DNA
Output: A desired DNA
Inserted part
Step 1: Cut given DNA using primers a and b
mutagenic primer a
Product A
mutagenic primer b
Product B
Step 2: Annealing inserted sequence using primers c and d
mutagenic primer c
mutagenic primer d
Product C
Step 3: Ligation PCR with product A,B and C
Desired DNA
Inserted part

Fig. 1. An example of site-directed insertion mutagenesis by PCR. Given a DNA sequence and four predesigned primers a, b, c and d, two primers a and b lead the DNA sequence to break and extend into two products A and B under enzymatic reaction (Step 1). Two primers c and d complementarily bind to desired insertion region according to the overlapping region and extend into product C (Step 2). Then, the products A, B and C join together to create recombinant DNA that include the desired insertion (Step 3).

In formal language theory, the insertion of a string means adding a substring to a given string and deletion of a string means removing a substring. The insertions occurring in DNA strands are in some sense context-sensitive and Kari and Thierrin [13] modeled such bio-operations using *contextual insertions and deletions* [8,19]. A finite set of insertion-deletion rules, together with a finite set of axioms, can be viewed as a language generating device. Contextual insertion-deletion systems in the study of molecular computing have been used e.g. by Daley et al. [3], Krassovitskiy et al. [14] and Takahara and Yokomori [21]. Further theoretical studies on the computational power of insertion-deletion systems were done e.g. by Margenstern et al. [17] and Păun et al. [18]. Enaganti et al. [6] have studied related operations to model the action of DNA polymerase enzymes.

We formalize site-directed insertion mutagenesis by PCR and define a new operation *outfix-guided insertion* that *partially* inserts a string y into a string x when two non-empty substrings of x match with an outfix of y, see Fig. 2(b). The outfix-guided insertion is an overlapping variant of the ordinary insertion operation, analogously as the overlap assembly [2,4,5], cf. Fig. 2(a), is a variant of the ordinary string concatenation operation.

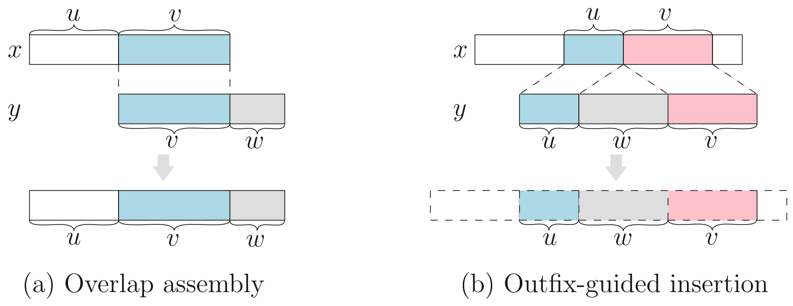

Fig. 2. (a) If suffix v of x overlaps with the prefix v of y, then the overlap assembly operation partially catenates x and y appending suffix w of y to x. (b) If the outfix of y consisting of u and v matches the substring uv of x, then the outfix-guided insertion operation inserts w between u and v in the string x.

This paper investigates the language theoretic closure properties of outfix-guided insertion and iterated outfix-guided insertion. Note that since outfix-guided insertion, similarly as overlap assembly, is not associative, there are more than one way to define the iteration of the operation. We consider a general outfix-guided insertion closure of a language which is defined analogously as the iterated overlap assembly by Enaganti et al. [4]. Iterated (overlap) assembly is defined by Csuhaj-Varju et al. [2] in a different way, which we call right one-sided iteration of an operation.

It is fairly easy to see that regular languages are closed under outfix-guided insertion. Closure of regular languages under outfix-guided insertion closure turns out to be less obvious. It is well known that regular languages are not closed under the iteration of the ordinary (non-overlapping) insertion operation [12]. However, the known counter-examples, nor their variants, do not work for iterated outfix-guided insertion. Here using a more involved construction we show that there exists even a finite language L such that the outfix-guided insertion closure of L is nonregular. On the other hand, we show that the outfix-guided insertion closure of a unary regular language is always regular.

It is well known that context-free languages are closed under ordinary (non-iterated) insertion. We show that context-free languages are not closed under outfix-guided insertion. The outfix-guided insertion of a regular language into a context-free language (or vice versa) is always context-free. Also we establish that a similar closure property does not hold for the deterministic context-free and the regular languages. Finally in the last section we consider decision problems on whether a language is closed under outfix-guided insertion (or og-closed). We give a polynomial time algorithm to decide whether a language recognized by a deterministic finite automaton (DFA) is og-closed. We show that for a given context-free language L the question of deciding whether or not L is og-closed is undecidable. Most proofs are omitted in this extended abstract.

2 Definition of (Iterated) Outfix-Guided Insertion

We assume the reader to be familiar with the basics of formal languages, in particular, with the classes of regular languages and (deterministic) context-free languages [20,22]. More details on variants of the insertion operation and iterated insertion can be found in [12].

The symbol Σ stands always for a finite alphabet, Σ^* is the set of strings over Σ, $|w|$ is the length of a string $w \in \Sigma^*$, w^R is the reversal of w and ε is the empty string. For $i \in \mathbb{N}$, $\Sigma^{\geq i}$ is the set of strings of length at least i.

If $w = xy$, $x, y \in \Sigma^*$, we say that x is a prefix of w and y is a suffix of w. If $w = xyz$, $x, y, z \in \Sigma^*$, we say that (x, z) is an *outfix* of w. If additionally $x \neq \varepsilon$ and $z \neq \varepsilon$, (x, z) is a *non-trivial outfix* of w. Sometimes (in particular, when talking about the outfix-guided insertion operation) we refer to an outfix (x, z) simply as a string xz (when it is known from the context what are the components x and z). For example, with $\Sigma = \{a, b, c\}$ and $w = abca$ the non-trivial outfixes of w are aa, aba, aca and $abca$.

We begin by recalling some notions associated with the non-overlapping insertion operation.[1] The non-overlapping insertion of a string y into a string x is defined as the set of strings $x \overset{\text{nol}}{\leftarrow} y = \{x_1 y x_2 \mid x = x_1 x_2\}$. The insertion operation is extended in the natural way for languages by setting $L_1 \overset{\text{nol}}{\leftarrow} L_2 = \bigcup_{x \in L_1, y \in L_2} x \overset{\text{nol}}{\leftarrow} y$. Following Kari [12] we define the *left-iterated insertion* of L_2 into L_1 inductively by setting

$$\mathbb{LI}^{(0)}(L_1, L_2) = L_1 \text{ and } \mathbb{LI}^{(i+1)}(L_1, L_2) = \mathbb{LI}^{(i)}(L_1, L_2) \overset{\text{nol}}{\leftarrow} L_2,\ i \geq 0.$$

[1] We use the term "non-overlapping" to make the distinction clear to outfix-guided insertion which will be the main topic of this paper.

The *left-iterated insertion closure of* L_2 *into* L_1 is $\mathbb{LI}^*(L_1, L_2) = \bigcup_{i=0}^{\infty} \mathbb{LI}^{(i)}(L_1, L_2)$. It is well known that the iterated non-overlapping insertion operation does not preserve regularity [10,12]. The left-iterated insertion closure of the string ab into itself is nonregular because $\mathbb{LI}^*(ab, ab) \cap a^*b^* = \{a^ib^i \mid i \geq 0\}$.

Next we define the main notion of this paper which can be viewed as a generalization of the overlap assembly operation [2,4]. The "inside part" of a string y can be outfix-guided inserted into a string x if a non-trivial outfix of y overlaps with a substring of x in a position where the insertion occurs. This differs from contextual insertion (as defined in [13]) in the sense that y must actually contain the outfix that is matched with a substring of x (and additionally [13] specifies a set of contexts where an insertion can occur).

Definition 1. *The* outfix-guided insertion *of a string y into a string x is defined as*

$$x \overset{\text{ogi}}{\leftarrow} y = \{x_1uzvx_2 \mid x = x_1uvx_2,\ y = uzv,\ u \neq \varepsilon, v \neq \varepsilon\}.$$

Using the above notations, when $x_1uzvx_2 \in x \overset{\text{ogi}}{\leftarrow} y$ we say that the nonempty substrings u and v are the *matched parts.* Note that the matched parts form a non-trivial outfix of the inserted string y.

Since we are almost exclusively dealing with outfix-guided insertion, in the following for notational simplicity we write just $\leftarrow$ in place of $\overset{\text{ogi}}{\leftarrow}$. Outfix-guided insertion is extended in the usual way for languages by setting $L_1 \leftarrow L_2 = \bigcup_{w_i \in L_i, i=1,2} w_1 \leftarrow w_2$.

Example 2. Outfix-guided insertion is not associative. Let $\Sigma = \{a, b, c, d\}$. Now $abcd \in (acd \leftarrow abc) \leftarrow abcd$ but $abc \leftarrow abcd = \emptyset$.

Since outfix-guided insertion is non-associative we define the $(i+1)$th iterated operation (analogously as was done with iterated overlap assembly [4]) by inserting to a string of the ith iteration another string of the ith iteration.

Definition 3. *For a language L define inductively*

$$\mathbb{OGI}^{(0)}(L) = L, \quad \textit{and}\ \mathbb{OGI}^{(i+1)}(L) = \mathbb{OGI}^{(i)}(L) \leftarrow \mathbb{OGI}^{(i)}(L),\ \ i \geq 0.$$

The outfix-guided insertion closure *of L is* $\mathbb{OGI}^*(L) = \bigcup_{i=0}^{\infty} \mathbb{OGI}^{(i)}(L)$.

For talking about specific iterated outfix-guided insertions, we use the notation $x \overset{[y]}{\Rightarrow} z$ to indicate that string z is in $x \leftarrow y$, $x, y, z \in \Sigma^*$. A sequence of steps

$$x \overset{[y_1]}{\Rightarrow} z_1 \overset{[y_2]}{\Rightarrow} z_2 \overset{[y_3]}{\Rightarrow} \cdots \overset{[y_m]}{\Rightarrow} z_m,\ \ m \geq 1,$$

is called a derivation of z_m from x.

When we want to specify the matched substrings, they are indicated by underlining. If $x = x_1uvx_2$ derives z by inserting uzv (where u and v are the matched prefix and suffix, respectively,) this is denoted

$$x_1\underline{uv}x_2 \overset{\underline{u}y\underline{v}}{\Rightarrow} z.$$

Also, sometimes underlining is done only in the inserted string if this makes it clear what must be the matched substrings in the original string.

By a *trivial derivation step* we mean a derivation $x \overset{[x]}{\Rightarrow} x$ where x is obtained from itself by selecting the outfix to consist of the entire string x. Every string of length at least two can be obtained from itself using a trivial derivation step. This means, in particular, that for any language L, $L - (\Sigma \cup \{\varepsilon\}) \subseteq \mathbb{OGI}^{(1)}(L)$. The sets $\mathbb{OGI}^{(i)}(L)$, $i \geq 1$, cannot contain strings of length less than two and, consequently $\mathbb{OGI}^{(i)}(L) \subseteq \mathbb{OGI}^{(i+1)}(L)$, for all $i \geq 1$.

Definition 3 iterates the outfix-guided insertion by inserting a string from the ith iteration of the operation into another string in the ith iteration. Since the operation is non-associative we can define iterated insertion in more than one way. The right one-sided insertion of L_2 into L_1 outfix-guided inserts a string of L_2 into L_1 and the iteration of the operation inserts a string obtained in the process into L_1. The iterated left one-sided outfix-guided insertion is defined symmetrically. In fact, when considering iterated ordinary insertion, Kari [12] uses a definition that we call left one-sided iterated insertion (and the operation was defined as $\mathbb{LI}^*(L_1, L_2)$ above). Csuhaj-Varju et al. [2] define iterated overlap assembly using right one-sided iteration of the operation.

Definition 4. Let L_1 and L_2 be languages. The *right one-sided iterated insertion of* L_2 *into* L_1 is defined inductively by setting $\mathbb{ROGI}^{(0)}(L_1, L_2) = L_2$ and $\mathbb{ROGI}^{(i+1)}(L_1, L_2) = L_1 \leftarrow \mathbb{ROGI}^{(i)}(L_1, L_2)$, $i \geq 0$. The *right one-sided insertion closure* of L_2 into L_1 is $\mathbb{ROGI}^*(L_1, L_2) = \bigcup_{i=0}^{\infty} \mathbb{ROGI}^{(i)}(L_1, L_2)$.

The *left one-sided iterated insertion of* L_2 *into* L_1 is defined inductively by setting $\mathbb{LOGI}^{(0)}(L_1, L_2) = L_1$ and $\mathbb{LOGI}^{(i+1)}(L_1, L_2) = \mathbb{LOGI}^{(i)}(L_1, L_2) \leftarrow L_2$, $i \geq 0$. The *left one-sided insertion closure* of L_2 into L_1 is $\mathbb{LOGI}^*(L_1, L_2) = \bigcup_{i=0}^{\infty} \mathbb{LOGI}^{(i)}(L_1, L_2)$.

The one-sided iterated insertion closures are defined for two argument languages. Naturally it would be possible to extend also the definition of unrestricted iterated outfix-guided insertion for two arguments. Note that for any language L, $\mathbb{OGI}^{(1)}(L) = \mathbb{LOGI}^{(1)}(L, L) = \mathbb{ROGI}^{(1)}(L, L) = L \leftarrow L$. On the other hand, the iterated version of unrestricted outfix-guided insertion is considerably more general than the one-sided variants. For any language L, $\mathbb{ROGI}^*(L, L)$ and $\mathbb{LOGI}^*(L, L)$ are always included in $\mathbb{OGI}^*(L)$ and, in general, the inclusions can be strict.

Example 5. Let $\Sigma = \{a, b, c\}$ and $L_1 = \{aacc\}$, $L_2 = \{abc\}$. Now $\mathbb{ROGI}^*(L_1, L_2) = a^+bc^+$. For example, by inserting abc into $aacc$ derives $aabcc$:

$$a\underline{ac}c \overset{\underline{a}b\underline{c}}{\Rightarrow} aabcc. \tag{1}$$

A right one-sided iterated insertion of L_2 into L_1 could then be continued, for example, as $a\underline{acc} \overset{\underline{a}ab\underline{cc}}{\Rightarrow} aaabcc$. In this way right one-sided derivations can generate all strings of a^+bc^+. Since all inserted strings must contain the symbol b, the first matched part must always belong to a^+ and the second matched part must belong to c^+. This means that $\mathbb{ROGI}^*(L_1, L_2) \subseteq a^+bc^+$.

On the other hand, $\mathbb{LOGI}^*(L_1, L_2) = \{aabcc, aacc\}$. In a left one-sided iterated insertion of L_2 into L_1, the only non-trivial derivation step is (1).

By denoting $L_3 = L_1 \cup L_2$, it can be verified that

$$\mathbb{OGI}^*(L_3) = \mathbb{ROGI}^*(L_3, L_3) = \mathbb{LOGI}^*(L_3, L_3) = a^+bc^+ \cup a^2a^*c^2c^*.$$

The next example illustrates that unrestricted outfix-guided insertion closure of a language L' can be larger than $\mathbb{LOGI}^*(L', L')$. The language L used in the proof of Theorem 9 in the next section gives an example where the unrestricted insertion closure is larger than $\mathbb{ROGI}^*(L, L)$ (as explained before Proposition 15).

Example 6. Let $\Sigma = \{a, b, c, d, e, f\}$ and $L' = \{abce, bcde, acdef\}$. We note that $a\underline{bce} \overset{\underline{bcde}}{\Rightarrow} abcde$. Furthermore, it is easy to verify that by outfix-guided inserting strings of L' into $L' \cup \{abcde\}$ one cannot produce more strings and, thus, $\mathbb{LOGI}^*(L', L') = L' \cup \{abcde\}$. On the other hand, we have

$$\underline{acde}f \overset{\underline{abcde}}{\Rightarrow} abcdef \in \mathbb{OGI}^{(2)}(L').$$

3 Outfix-Guided Insertion and Regular Languages

As can be expected, the family of regular languages is closed under outfix-guided insertion. On the other hand, the answer to the question whether regular languages are closed under outfix-guided insertion closure seems less clear. From Kari [12] we recall that it is easy to construct examples that establish the nonclosure of regular languages under iterated non-overlapping insertion. However, such straightforward counter-examples do not work for the unrestricted outfix-guided insertion closure. Using a more involved construction we establish that even the outfix-guided insertion closure of a finite language need not be regular. The nonclosure of regular languages under right one-sided insertion closure is established by a more straightforward construction.

Proposition 7. *If L_1 and L_2 are regular, then so is $L_1 \leftarrow L_2$.*

It seems difficult to extend the proof of Proposition 7 for outfix-guided insertion closure because on strings with iterated insertions, the computations on corresponding prefix-suffix pairs can, in general, depend on each other and when processing a part inserted in between, an NFA would need to keep track of such pairs, as opposed to simply keep track of a set of states. On the other hand, it is not equally easy as in the case of non-overlapping iterated insertion to construct a counter-example, i.e., a regular language whose outfix-guided insertion closure is nonregular.

Next we show that regular languages, indeed, are not closed under iterated outfix-guided insertion. For the construction we use the following technical lemma.

Lemma 8. *Let $\Sigma = \{a_1, a_2, a_3, b_1, b_2, b_3\}$ and define*

$$L_1 = \{a_3a_1a_2b_1,\ \ a_2b_2b_1b_3,\ \ a_1a_2a_3b_2,\ \ a_3b_3b_2b_1,\ \ a_2a_3a_1b_3,\ \ a_1b_1b_3b_2\}.$$

Then $L_1 \leftarrow L_1 = L_1$.

Theorem 9. *There exists a finite language L such that $\mathbb{OGI}^*(L)$ is nonregular.*

Proof *(Sketch).* Let $\Sigma = \{a_1, a_2, a_3, b_1, b_2, b_3\}$ and define

$$L = \{\$a_3a_1b_1b_3\$,\ a_3a_1a_2b_1,\ a_2b_2b_1b_3,\ a_1a_2a_3b_2,\ a_3b_3b_2b_1,\ a_2a_3a_1b_3,\ a_1b_1b_3b_2\}.$$

Note that $L - \{\$a_3a_1b_1b_3\$\}$ is equal to the language L_1 from Lemma 8. For ease of discussion we introduce names for the strings of L_1: $y_1 = a_3a_1a_2b_1$, $y_2 = a_2b_2b_1b_3$, $y_3 = a_1a_2a_3b_2$, $y_4 = a_3b_3b_2b_1$, $y_5 = a_2a_3a_1b_3$, $y_6 = a_1b_1b_3b_2$. and define the finite set

$$S_{\text{middle}} = \{a_1b_1,\ a_1a_2b_1, a_1a_2b_2b_1,\ a_1a_2a_3b_2b_1,\ a_1a_2a_3b_3b_2b_1,\ a_1a_2a_3a_1b_3b_2b_1\}.$$

We claim that

$$\mathbb{OGI}^*(L) = \{\$a_3(a_1a_2a_3)^iz(b_3b_2b_1)^ib_3\$ \mid i \geq 0,\ z \in S_{\text{middle}}\}. \tag{2}$$

To establish the inclusion from right to left, we note that

$$\$a_3a_1b_1b_3\$ \overset{[y_1]}{\Rightarrow} \$a_3a_1a_2b_1b_3\$ \overset{[y_2]}{\Rightarrow} \$a_3a_1a_2b_2b_1b_3\$ \overset{[y_3]}{\Rightarrow} \$a_3a_1a_2a_3b_2b_1b_3\$ \overset{[y_4]}{\Rightarrow}$$
$$\$a_3a_1a_2a_3b_3b_2b_1b_3\$ \overset{[y_5]}{\Rightarrow} \$a_3a_1a_2a_3a_1b_3b_2b_1b_3\$ \overset{[y_6]}{\Rightarrow} \$a_3a_1a_2a_3a_1b_1b_3b_2b_1b_3\$ = w_1.$$

The first five insertions generate the strings $\$a_3zb_3\$$, $z \in S_{\text{middle}}$, and the last string w_1 again has "middle part" $a_3a_1b_1b_3$. By cyclically outfix-guided inserting the strings $y_1, \ldots y_6$ into w_1 we get all strings $\$a_3(a_1a_2a_3)z(b_3b_2b_1)b_3\$$, $z \in S_{\text{middle}}$, and the string $\$a_3(a_1a_2a_3)^2a_1b_1(b_3b_2b_1)^2b_3\$$. By simple induction it follows that $\mathbb{OGI}^*(L)$ contains the right side of (2).

To establish the converse inclusion, we verify using Lemma 8 that all strings obtained by iterated outfix-guided insertion from strings of L must be obtained as above, that is, all non-trivial derivations producing new strings must be as above. ■

We conjecture that the iterated outfix-guided insertion closure of a regular language need not be even context-free. However, a construction of such a language would seem to be considerably more complicated than the construction used in the proof of Theorem 9.

Contrasting the result of Theorem 9 we show that unary regular languages are closed under iterated outfix-guided insertion. The construction is based on a technical lemma which shows that, for unary languages, outfix-guided insertion closure can be represented as a variant of the iterated overlap assembly [2,4].

Definition 10. Let $x, y \in \Sigma^*$. The *2-overlap catenation* of x and y, $x\overline{\odot}^2 y$, is defined as

$$x\overline{\odot}^2 y = \{z \in \Sigma^+ \mid (\exists u, w \in \Sigma^*)(\exists v \in \Sigma^{\geq 2})\ x = uv, y = vw, z = uvw\}.$$

For $L \subseteq \Sigma^*$, we define inductively $2\mathbb{OC}^{(0)}(L) = L$ and $2\mathbb{OC}^{(i+1)}(L) = 2\mathbb{OC}^{(i)}(L)\overline{\odot}^2 2\mathbb{OC}^{(i)}(L)$, $i \geq 0$. The *2-overlap catenation closure* of L is $2\mathbb{OC}^*(L) = \bigcup_{i=0}^{\infty} 2\mathbb{OC}^{(i)}(L)$.

Due to commutativity of unary languages we get the following property which will be crucial for establishing closure of unary regular languages under outfix-guided insertion closure.

Lemma 11. *If $x, y \in a^*$ are unary strings, then $x \leftarrow y = x\overline{\odot}^2 y$.*

Corollary 12. *If L is a unary language then $\mathbb{OGI}^*(L) = 2\mathbb{OC}^*(L)$.*

The 2-overlap closure of a regular language is always regular. The construction does not depend on a language being unary, so we state the result for regular languages over an arbitrary alphabet. Csuhaj-Varju et al. [2] have shown that iterated overlap assembly preserves regularity. The proof of Lemma 13 is inspired by Theorem 4 of [2] but does not follow from it because [2] defines iteration of operations as right one-sided iteration and, furthermore, 2-overlap catenation has an additional length restriction on the overlapping strings.

Lemma 13. *The 2-overlap catenation closure of a regular language is regular.*

By Corollary 12 and Lemma 13 we have shown that unary regular languages are closed under outfix-guided insertion closure, contrasting the result of Theorem 9 for general regular languages.

Theorem 14. *The outfix-guided insertion closure of a unary regular language is always regular.*

The left and right one-sided insertion closures are restricted variants of the general outfix-guided insertion closure, so Theorem 9 does not, at least not directly, imply the existence of regular languages L_1 and L_2 such that $\mathbb{LOGI}^*(L_1, L_2)$ or $\mathbb{ROGI}^*(L_1, L_2)$ are non-regular. Here we show that the one-sided outfix-guided insertion closures are not, in general, regularity preserving. For left-one one-sided outfix-guided insertion closure the construction is similar to that used in the proof of Theorem 9. However, this construction does not work for right one-sided closure because if L is the language used in the proof of Theorem 9, then $\mathbb{ROGI}^*(L, L)$ is the finite language $L \cup \{\$a_3a_1a_2b_1b_3\$\}$.

Proposition 15. *There exist finite languages L_1, L_2, L_3 and L_4 such that $\mathbb{ROGI}^*(L_1, L_2)$ and $\mathbb{LOGI}^*(L_3, L_4)$ are non-regular.*

4 Outfix-Guided Insertion and Context-Free Languages

It is well known that the family of context-free languages is closed under ordinary insertion. Contrasting the result of Proposition 7 we show that context-free languages are not closed under (non-iterated) outfix-guided insertion.

Theorem 16. *There exists a context-free language L such that $L \leftarrow L$ is not context-free.*

It follows that context-free languages are not closed under one-sided outfix-guided iteration because, for any language L, $\mathbb{OGI}^{(1)}(L) = \mathbb{ROGI}^{(1)}(L, L) = \mathbb{LOGI}^{(1)}(L, L) = L \leftarrow L$. On the other hand, the outfix-guided insertion of a regular (respectively, context-free) language into a context-free (respectively, regular) language is always regular.

Theorem 17. *If L_1 is context-free and L_2 is regular, then $L_1 \leftarrow L_2$ and $L_2 \leftarrow L_1$ are context-free.*

The analogy of Theorem 17 does not hold for deterministic context-free languages. Techniques for proving that a language is not deterministic context-free are known already from [9].

Theorem 18. *If L_1 is deterministic context-free and L_2 is regular, the languages $L_1 \leftarrow L_2$ or $L_2 \leftarrow L_1$ need not be deterministic context-free.*

Theorem 16 raises the question how complex languages can be obtained from context-free languages using iterated outfix-guided insertion. Note that if L_1 and L_2 are context-free, it is easy to verify that $L_1 \leftarrow L_2$ is at least deterministic context-sensitive.

Proposition 19. *If L_1 and L_2 are context-free then $\mathbb{ROGI}^*(L_1, L_2)$ and $\mathbb{LOGI}^*(L_1, L_2)$ are context-sensitive.*

In the proof of Proposition 19 it is sufficient to know that the languages L_1 and L_2 are context-sensitive, and as a consequence it follows that context-sensitive languages are closed under one-sided outfix-guided insertion closure.

Corollary 20. *If L_1 and L_2 are context-sensitive then so are $\mathbb{ROGI}^*(L_1, L_2)$ and $\mathbb{LOGI}^*(L_1, L_2)$.*

We conjecture that, for any context-free language L, $\mathbb{OGI}^*(L)$ must be context-sensitive. Constructing a linear bounded automaton for $\mathbb{OGI}^*(L)$ is more difficult than in the case of the right or left one-sided insertion closures, because a direct simulation of a derivation of $w \in \mathbb{OGI}^*(L)$ (i.e., simulation of the iterated outfix-guided insertion steps producing w) would need to remember, at a given time, an unbounded number of substrings of the input.

Also we do not know how to make the procedure in the proof of Proposition 19 deterministic and it remains open whether the one-sided outfix-guided insertion closures of context-free languages are always deterministic context-sensitive.

5 Deciding Closure Under Outfix-Guided Insertion

We say that a language L is *closed* under outfix-guided insertion, or *og-closed* for short, if outfix-guided inserting strings of L into L does not produce strings outside of L, that is, $(L \leftarrow L) \subseteq L$.

A natural algorithmic problem is then to decide for a given language L whether or not L is og-closed. If L is regular, by Proposition 7, we can decide whether or not L is og-closed. For a given DFA A, Proposition 7 yields only an NFA for the language $L(A) \leftarrow L(A)$. In general, the NFA equivalence or inclusion problem is PSPACE complete [22], however, inclusion of an NFA language in the language $L(A)$ can be tested efficiently when A is deterministic.

Proposition 21. *There is a polynomial time algorithm to decide whether for a given DFA A the language $L(A)$ is og-closed.*

The method used in Proposition 21 does not yield an efficient algorithm if the regular language L is specified by an NFA. The complexity of deciding og-closure of a language accepted by an NFA remains open. On the other hand, using a reduction from the Post Correspondence Problem it follows that the question whether or not a context-free language is og-closed in undecidable.

Theorem 22. *For a given context-free language L, the question whether or not L is og-closed is undecidable.*

6 Conclusion

Analogously with the recent overlap assembly operation [2,4], we have introduced an overlapping insertion operation on strings and have studied closure and decision properties of the outfix-guided insertion operation. While closure properties of non-iterated outfix-guided insertion are straightforward to establish, the questions become more involved for the outfix-guided insertion closure. As the main result we have shown that the outfix-guided insertion closure of a finite language need not be regular.

Much work remains to be done on outfix-guided insertion. One of the main open questions is to determine upper bounds for the complexity of the outfix-guided insertion closures of regular languages. Does there exist regular languages L such that the outfix-guided insertion closure of L is non-context-free?

Acknowledgments. Cho and Han were supported by the Basic Science Research Program through NRF funded by MEST (2015R1D1A1A01060097), the Yonsei University Future-leading Research Initiative of 2015 and the International Cooperation Program managed by NRF of Korea (2014K2A1A2048512). Ng and Salomaa were supported by Natural Sciences and Engineering Research Council of Canada Grant OGP0147224.

References

1. Bertram, J.S.: The molecular biology of cancer. Mol. Asp. Med. **21**(6), 167–223 (2000)
2. Csuhaj-Varju, E., Petre, I., Vaszil, G.: Self-assembly of strings and languages. Theoret. Comput. Sci. **374**, 74–81 (2007)
3. 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)
4. Enaganti, S., Ibarra, O., Kari, L., Kopecki, S.: On the overlap assembly of strings and languages. Nat. Comput. (2016). dx.doi.org/10.1007/s11047-015-9538-x
5. Enaganti, S.K., Ibarra, O.H., Kari, L., Kopecki, S.: Further remarks on DNA overlap assembly, manuscript (2016)
6. Enaganti, S.K., Kari, L., Kopecki, S.: A formal language model of dna polymerase enzymatic activity. Fundam. Inform. **138**, 179–192 (2015)
7. Flavell, R., Sabo, D., Bandle, E., Weissmann, C.: Site-directed mutagenesis: effect of an extracistronic mutation on the in vitro propagation of bacteriophage qbeta RNA. Proc. Natl. Acad. Sci. **72**(1), 367–371 (1975)
8. Galiukschov, B.: Semicontextual grammars (in Russian). Mat. Log. Mat. Lingvistika 38–50 (1981)
9. Ginsburg, S., Greibach, S.: Deterministic context free languages. Inf. Control **9**, 620–648 (1966)
10. Haussler, D.: Insertion languages. Inf. Sci. **31**, 77–89 (1983)
11. Hemsley, A., Arnheim, N., Toney, M.D., Cortopassi, G., Galas, D.J.: A simple method for site-directed mutagenesis using the polymerase chain reaction. Nucleic Acids Res. **17**(16), 6545–6551 (1989)
12. Kari, L.: On insertion and deletion in formal languages. Ph.D. thesis, University of Turku (1991)
13. Kari, L., Thierrin, G.: Contextual insertions/deletions and computability. Inf. Comput. **131**(1), 47–61 (1996)
14. Krassovitskiy, A., Rogozhin, Y., Verlan, S.: Computational power of insertion-deletion (P) systems with rules of size two. Nat. Comput. **10**, 835–852 (2011)
15. Lee, J., Shin, M.K., Ryu, D.K., Kim, S., Ryu, W.S.: Insertion and deletion mutagenesis by overlap extension PCR. In: Braman, J. (ed.) In Vitro Mutagenesis Protocols, 3rd edn, pp. 137–146. Humana Press, New York (2010)
16. Liu, H., Naismith, J.H.: An efficient one-step site-directed deletion, insertion, single and multiple-site plasmid mutagenesis protocol. BMC Biotechnol. **8**(1), 91–101 (2008)
17. Margenstern, M., Păun, G., Rogozhin, Y., Verlan, S.: Context-free insertion-deletion systems. Theoret. Comput. Sci. **330**(2), 339–348 (2005)
18. Păun, G., Pérez-Jiménez, M.J., Yokomori, T.: Representations and characterizations of languages in Chomsky hierarchy by means of insertion-deletion systems. Int. J. Found. Comput. Sci. **19**(4), 859–871 (2008)
19. Păun, G.: On semicontextual grammars. Bull. Math. Soc. Sci. Math. Rouman. **28**, 63–68 (1984)
20. Shallit, J.: A Second Course in Formal Languages and Automata Theory. Cambridge University Press, Cambridge (2009)
21. Takahara, A., Yokomori, T.: On the computational power of insertion-deletion systems. Nat. Comput. **2**, 321–336 (2003)
22. Yu, S.: Regular languages. In: Salomaa, A., Rozenberg, G. (eds.) Handbook of Formal Languages, vol. I, pp. 41–110. Springer, Heidelberg (1997)