Fundamenta Informaticae 72 (2006) 1–14
IOS Press

Block Substitutions and Their Properties

Lila Kari and Elena Losseva*
Department of Computer Science
The University of Western Ontario
London, ON, Canada, N6A 5B7
{*lila,elena*}*@csd.uwo.ca*

Abstract. We introduce a type of substitution operation inspired by errors occurring in biologically encoded information. We derive the closure properties of language families in the Chomsky hierarchy under these substitution operations. Moreover, we consider some language equations involving these operations and investigate decidability of the existence of solutions to such equations.

Keywords: Formal languages, Chomshy hierarchy, block substitution

1. Introduction

The subject of language operations is a classical topic in formal language theory [15], [7]. Some of the related questions involve closure properties of language families in the Chomsky hierarchy under such operations, as well as related language equations. Language equations involving the catenation operation have been introduced and investigated in [2]. Language equations involving operations other than catenation have been studied in [8], [11]. More recently, another language operation has been studied in the context of error detection and error correction of information transmitted via a noisy channel. The substitution operation represents a single error by a substitution of one character in a word by another character. In this paper, a different type of substitution is studied. The difference is twofold. On one hand, the substitution occurs only in a single location, i.e., only one subword of a word can be replaced by a word of equal length. On the other hand, the substituted letters need not be distinct from the original ones. This modification was inspired by the phenomenon of errors occurring in DNA strands (mutations, insertions, deletions), wherein not all substituted nucleotides need to be different from the original ones.

*Address for correspondence: Department of Computer Science, The University of Western Ontario, London, ON, Canada, N6A 5B7

The paper is structured as follows. First, the new word operations are introduced, together with their inverses. Then, the closure properties of language families in the Chomsky hierarchy under these operations are studied. The paper concludes with the study of language equations involving these operations.

2. Preliminary Definitions and Notation

A *binary word operation* is a mapping $\diamondsuit : \Sigma^* \times \Sigma^* \to 2^{\Sigma^*}$, where 2^{Σ^*} is the set of all subsets of Σ^*.

For any languages X and Y, $X \diamondsuit Y = \bigcup_{u \in X, v \in Y} u \diamondsuit v$.

Definition 1. ([8]) Let $\diamondsuit$ be an operation. The left inverse $\diamondsuit^l$ of $\diamondsuit$ is defined as

$$w \in (x \diamondsuit v) \text{ iff } x \in (w \diamondsuit^l v), \text{ for all } v, x, w \in \Sigma^*,$$

and the right inverse $\diamondsuit^r$ of $\diamondsuit$ is defined as

$$w \in (u \diamondsuit y) \text{ iff } y \in (u \diamondsuit^r w), \text{ for all } u, y, w \in \Sigma^*.$$

Definition 2. Let $\diamondsuit$ be a binary word operation. The word operation $\diamondsuit'$ defined by $u \diamondsuit' v = v \diamondsuit u$ is called reversed $\diamondsuit$.

If x and y are symbols in $\{l, r, '\}$, the notation $\diamondsuit^{xy}$ represents the operation $(\diamondsuit^x)^y$. Using the above observations, one can establish identities between operations of the form $\diamondsuit^{xy}$. For example, $\diamondsuit^{ll} = \diamondsuit^{rr} = \diamondsuit^{''} = \diamondsuit$ and $\diamondsuit'^l = \diamondsuit^{r'} = \diamondsuit^{lr}$.

Next we list a few binary word operations [7], [8].

Catenation:[1] $u \cdot v = \{uv\}$.

Shuffle (or scattered insertion): $u \amalg v = \{u_1 v_1 \cdots u_k v_k u_{k+1} \mid k \geq 1, u = u_1 \cdots u_k u_{k+1}, v = v_1 \cdots v_k\}$.

Language equations

The process of solving language equations has much in common with the process of solving algebraic equations. For example, the equation $X \diamondsuit L = R$ is similar to the equation $x + a = b$, where a, b are constants. In both cases, the unknown left operand can be obtained from the result of the operation and the known operand by using an "inverse" operation. In the case of addition, this role is played by subtraction. In the case of a binary word operation, which usually is not commutative, the notion of left inverse has to be utilized. Similarly, the notion of right-inverse will aid in solving equations of the type $L \diamondsuit Y = R$, where the unknown is the right-operand. We recall now a result from [8] that uses the left and right inverse operations to solve language equations.

Theorem 1. Let $L, R \subseteq \Sigma^*$ be two languages and let $\diamondsuit$ be a binary word operation. If the equation $X \diamondsuit L = R$ (respectively $L \diamondsuit Y = R$) has a solution, then the language $X_{max} = (R^c \diamondsuit^l L)^c$ (respectively $Y_{max} = (L \diamondsuit^r R^c)^c$) is also a solution, namely one that includes all the other solutions to the equation.

[1] We shall also write uv for $u \cdot v$.

Next we consider two natural binary word operations related to scattered substitutions [9].

Definition 3. If $u, v \in \Sigma^*$, then we define the *substitution in u by v* as

$u \bowtie v = \{u_1v_1u_2v_2 \dots u_kv_ku_{k+1} \mid k \geq 0, u = u_1a_1u_2a_2 \dots u_ka_ku_{k+1}, v = v_1v_2 \dots v_k, a_i, v_i \in \Sigma, 1 \leq i \leq k, a_i \neq v_i, \forall i, 1 \leq i \leq k\}$.

The case $k = 0$ corresponds to $v = \lambda$ when no substitution is performed.

Definition 4. If $u, v \in \Sigma^*$, then we define the *substitution in u of v* as

$u \triangle v = \{u_1a_1u_2a_2 \dots u_ka_ku_{k+1} \mid k \geq 0, u = u_1v_1u_2v_2 \dots u_kv_ku_{k+1}, v = v_1v_2 \dots v_k, a_i, v_i \in \Sigma, 1 \leq i \leq k, a_i \neq v_i, \forall i, 1 \leq i \leq k\}$.

Lemma 1. The operation $\bowtie$ is the left-inverse of $\triangle$.

Next we look at the right inverses of $\bowtie$ and $\triangle$.

Definition 5. For any words $u, v \in \Sigma^*$ of the same length and with Hamming distance $H(u, v) = k$, for some nonnegative integer k, $u \rhd v$ is the set of words

$$b_1b_2 \dots b_k,\ b_i \in \Sigma, 1 \leq i \leq k,$$

such that $u = u_1a_1 \cdots u_ka_ku_{k+1}$, $v = u_1b_1 \cdots u_kb_ku_{k+1}$ and, for all i, $1 \leq i \leq k$, $a_i \neq b_i$.

In other words, $u \rhd v$ consists of the word $b_1b_2 \cdots b_k$ where $b_1, b_2, \dots, b_k$ are the symbols of v that are different from the corresponding symbols of u. It should be clear that the set $u \rhd v$ is empty when u and v have different lengths.

Example 2.1. If $L_1 = \{a^nb^n \mid n \geq 1\}$ and $L_2 = \{b^m \mid m \geq 1\}$, then $L_1 \rhd L_2 = b^*$. (We can only perform $a^nb^n \rhd b^{2n}$ which gives b^n.) On the other hand, $L_2 \rhd L_1 = a^*$. Hence, the operation $\rhd$ is not commutative.

Note that $\rhd$ is the right inverse of $\bowtie$, and the reversed $\rhd$ is the right inverse of $\triangle$.

3. Block Substitution Operations and their Inverses

The substitution operation is a binary word operation, which is used to model an error in DNA-inspired computation. In this section we define three new block substitution operations. We also examine their left and right inverses.

Definition 6. If $u, v \in \Sigma^*$, then we define the *block substitution in u by v* as

$$u \bowtie_b v = \{u_1vu_3 \mid u = u_1u_2u_3, |u_2| = |v|, u_i \in \Sigma^*\}.$$

Unlike the $\bowtie$ operation, $\bowtie_b$ allows a letter to be replaced by any letter, not necessarily distinct. The following example demonstrates that $\bowtie_b$ is not a special case of the $\bowtie$ operation.

Example 3.1. Let $u = aaba$ and $v = bb$. Observe that the word $abba$ is in $u \bowtie_b v$ but not in $u \bowtie v$. On the other hand, the word $babb$ is in $u \bowtie v$, but it is not in $u \bowtie_b v$.

Definition 7. If $u, v \in \Sigma^*$, then we define the *block substitution of* v *in* u as

$$u \triangle_b v = \{u_1 \alpha u_2 \mid u = u_1 v u_2, |\alpha| = |v|, u_i, v \in \Sigma^*\}.$$

Again, we can demonstrate that $\triangle_b$ is not a special case of the $\triangle$ operation.

Example 3.2. Let $u = aabaaba$ and $v = aba$. Observe that the word $aaaaaba$ is in $u \triangle_b v$ but not in $u \triangle v$. On the other hand, the word $baaabba$ is in $u \triangle v$, but it is not in $u \triangle_b v$.

Proposition 1. The operation $\bowtie_b$ is the left inverse of $\triangle_b$.

Proof:
Let $w \in u \bowtie_b v$. Then $u = u_1 u_2 u_3$, $|u_2| = |v|, w = u_1 v u_3$ for some $u_i \in \Sigma^*, v \in \Sigma^*$. This means $u \in w \triangle_b v$.

Conversely, let $u \in w \triangle_b v$. Then $w = w_1 v w_2, |v| = |\alpha|, u = w_1 \alpha w_2$. This means $w \in u \bowtie_b v$. □

Definition 8. For any two words $u, v \in \Sigma^*$ of the same length define

$$u \rhd_b v = \{\beta \mid u = u_1 \alpha u_2, v = u_1 \beta u_2, |\alpha| = |\beta|\}.$$

When $|u| \neq |v|$ the set $u \rhd_b v$ is empty.

Proposition 2. The operation $\rhd_b$ is the right inverse of $\bowtie_b$.

Proof:
Let $w \in u \bowtie_b y$. Then $u = u_1 u_2 u_3$, $|u_2| = |y|, w = u_1 y u_3$ for some $u_i \in \Sigma^*, y \in \Sigma^*$. This means $y \in u \rhd_b w$.

Conversely, let $y \in u \rhd_b w$. Then $u = u_1 \alpha u_2, |y| = |\alpha|, w = u_1 y u_2$. This means $w \in u \bowtie_b v$. □

The next example shows that $\rhd_b$ is not a special case of the $\rhd$ operation.

Example 3.3. Let $u = aaaaa$ and $v = ababa$. Observe that the word bab is in $u \rhd_b v$ but not in $u \rhd v$. On the other hand, the word bb is in $u \rhd v$, but not in $u \rhd_b v$.

We now show that left and right inverses of all related operations are as shown in Table 1.

Proposition 3. The operation $\rhd'_b$ is the right inverse of $\triangle_b$.

Proof:
Let $w \in u \triangle_b y$. Then $u = u_1 y u_2, w = u_1 \alpha u_2, |\alpha| = |y|$. This means that $y \in (w \rhd_b u)$, hence $y \in (u \rhd'_b w)$. Conversely, suppose that $y \in (u \rhd'_b w)$. Then $y \in (w \rhd_b u), w = w_1 \alpha w_2, u = w_1 y w_2, |\alpha| = |y|$. This means that $w \in (u \triangle_b y)$. □

Proposition 4. The operation $\triangle'_b$ is the left inverse of $\rhd_b$.

Proof:
Suppose $w \in (u \rhd_b v)$. Then $w = \beta, u = u_1 \alpha u_2, v = u_1 \beta u_2, |\alpha| = |\beta|$. Hence $u \in (v \triangle_b w)$ and $u \in (w \triangle'_b v)$. Conversely, suppose $u \in (w \triangle'_b v)$. Then $u \in (v \triangle_b w), v = v_1 w v_2, u = v_1 \alpha v_2, |\alpha| = |w|$. This means that $w \in (u \rhd_b v)$. □

operation $\diamondsuit$	left inverse of $\diamondsuit$	right inverse of $\diamondsuit$
$\bowtie_b$	$\triangle_b$	$\triangleright_b$
$\triangle_b$	$\bowtie_b$	$\triangleright'_b$
$\triangleright_b$	$\triangle'_b$	$\bowtie_b$
$\triangleright'_b$	$\bowtie'_b$	$\triangle_b$
$\bowtie'_b$	$\triangleright'_b$	$\triangle'_b$
$\triangle'_b$	$\triangleright_b$	$\bowtie'_b$

Table 1. Substitution operations and their inverses

Proposition 5. The operation $\bowtie'_b$ is the left inverse of $\triangleright'_b$.

Proof:
$w \in (u \triangleright'_b v)$ iff $w \in (v \triangleright_b u)$ iff $u \in (v \triangleright_b w)$ iff $u \in (w \bowtie'_b v)$. □

Proposition 6. The right inverse of $\bowtie'_b$ is $\triangle'_b$.

Proof:
$w \in (u \bowtie'_b v)$ iff $w \in (v \bowtie_b u)$ iff $v \in (w \triangle_b u)$ iff $v \in (u \triangle'_b w)$. □

4. Closure Properties of Block Substitution Operations

The closure properties of language families in the Chomsky hierarchy under the operations of scattered insertion and deletion were first studied in [7]. In [9] the operations of scattered substitution were investigated. In this section we investigate such closure properties for the block substitution operations, namely $\bowtie_b, \triangle_b, \triangleright_b$.

Proposition 7. If L and R are languages over the alphabet Σ, R a regular one, then $L \triangle_b R$ is the image of L through a λ-free gsm.

Proof:
Let $A = (S, \Sigma, s_1, F, P)$ be an NFA that recognizes a regular language R over Σ. Construct the following gsm $g = (S', \Sigma, \Sigma, s_0, F', P')$, where

$$
\begin{aligned}
S' &= S \cup \{s_0, s_f\}, \text{ and } s_f, s_0 \notin S \cup F, \\
F' &= F \cup \{s_f\}, \\
P' &= \{s_0 a \to a s_0 \mid a \in \Sigma\} && (1)\\
&\cup \{s_0 a \to b s' \mid s_1 a \to s' \in P; a, b \in \Sigma\} && (2)\\
&\cup \{sa \to b s' \mid sa \to s' \in P; a, b \in \Sigma\} && (3)\\
&\cup \{s_f a \to a s_f \mid a \in \Sigma\} && (4)\\
&\cup \{sa \to a s_f \mid a \in \Sigma, s \in F\}. && (5)
\end{aligned}
$$

Recall that $u \triangle_b v$ is a block substitution of v in u by a word of equal length. The gsm g is constructed from NFA A in a way that allows each transition of A to replace an input character by any character of the alphabet, which is then outputted by the gsm. In addition, g has a new start state s_0 and an extra final state s_f, which are used to skip over any input prefix and/or suffix, outputting it without change. The production rules P' can be explained as follows. Rule (1) allows the gsm to output any prefix of its input without change. Rules (2) and (3) are responsible for recognizing a subword of its input equal to a word in R and substitution of that subword by an arbitrary word of the same length. Rules (4) and (5) allow the gsm to output any suffix of its input without change. □

Corollary 1. The REG and CF families of languages are closed under $\triangle_b$ with regular languages.

Proposition 8. CF is not closed under $\triangle_b$.

Proof:
There exist two context-free languages L_1 and L_2 such that $L_1 \triangle_b L_2$ is not context-free. Let $\Sigma = \{a, b, c, d, e\}$ and consider the two context-free languages over Σ

$$L_1 = \{a^n b^n c^m d^m \mid n, m \geq 1\},$$
$$L_2 = \{b^n c^n \mid n \geq 1\}.$$

Consider the language L defined as

$$L = (L_1 \triangle_b L_2) \cap a^* e^* d^*.$$

Observe that

$$L = \{a^{n+i} e^{2n-i-k} d^{m+k} \mid i, j, k \geq 1\},$$

which is not context-free. As CF is closed under intersection with regular languages, it follows that CF is not closed under $\triangle_b$. □

Proposition 9. The family of CS languages is closed under $\triangle_b$.

Proof:
Let L_1, L_2 be two context-sensitive languages over Σ and let $\Sigma' = \{a' \mid a \in \Sigma\}$, $\Sigma'' = \{a'' \mid a \in \Sigma\}$. Let the gsm g be such that it transforms a nonempty subword of its input into its primed version.

Formally, $g = (S, \Sigma, \Sigma \cup \Sigma', s_0, F, P)$, with $S = \{s_0, s_1, s_2\}$, $F = \{s_1, s_2\}$, and

$$\begin{aligned} P \;=\; & \{s_0 a \to a s_0, s_0 a \to a' s_1, s_1 a \to a' s_1, s_1 a \to a' s_2, \\ & s_1 a \to a s_2, s_2 a \to a s_2 \mid \forall a \in \Sigma\}. \end{aligned}$$

Let h be a morphism that changes each letter to its double-primed version, i.e., $h : \Sigma \to \Sigma'', h(a) = a'', a \in \Sigma$. Also, let $h' : \Sigma \cup \Sigma' \cup \Sigma'' \to \Sigma \cup \Sigma' \cup \{\lambda\}$ be a morphism that deletes all double-primed letters, i.e., $h'(a) = a, h'(a') = a', h'(a'') = \lambda$.

Let g' be a gsm that changes all primed letters of its input into any non-primed letters. Formally, $g' = (\{s_0\}, \Sigma \cup \Sigma', \Sigma, s_0, \{s_0\}, P)$, where

$$P = \{s_0 a \to a s_0 \mid \forall a \in \Sigma\} \cup \{s_0 a' \to b s_0 \mid \text{ for any } a, b \in \Sigma\}.$$

We claim that

$$L_1 \triangle_b L_2 = g'\{h'[[g(L_1) \amalg h(L_2)] \cap R]\},$$

where R is the regular language

$$R = \Sigma^*(\bigcup_{a\in\Sigma} a'a'')^*\Sigma^*.$$

Informally, R is the set of all strings where a primed letter is immediately followed by its double-primed version and the unprimed letters appear only in the prefixes and suffixes of the words. The intersection with R ensures that only words $g(u)$ and $h(v)$ are shuffled where v is a subword of u. Only words where a primed letter is followed by an identical double primed letter are kept. Applying h' erases the double primed letters, while g' replaces the primed subword with any word (of equal length) containing only unprimed letters.

The morphism h' is a 2-linear erasing morphism with respect to the language it is applied to as it erases at most half of each word. CS is closed under k-linear erasing as well as the other operators involved, hence CS is closed under $\triangle_b$. □

Proposition 10. If $L_1, L_2 \subseteq \Sigma^*$, L_2 regular, then $L_1 \bowtie_b L_2$ is the image of L_1 through a λ-free gsm.

Proof:
Let $A = (S, \Sigma, s, F, P)$ be an NFA that recognizes L_2 over Σ. Construct the gsm $g = (S', \Sigma, \Sigma, s_0, F', P')$, where

$$\begin{aligned}
S' &= S \cup \{s_0, s_f\}, \\
F' &= F \cup \{s_f\}, \\
P' &= \{s_0 a \to a s_0 \mid a \in \Sigma\} && (1)\\
&\cup \{s_0 b \to a s' \mid s_1 a \to s' \in P; a, b \in \Sigma\} && (2)\\
&\cup \{sb \to a s' \mid sa \to s' \in P; a, b \in \Sigma\} && (3)\\
&\cup \{s_f a \to a s_f \mid a \in \Sigma\} && (4)\\
&\cup \{sa \to a s_f \mid a \in \Sigma, s \in F\}. && (5)
\end{aligned}$$

The gsm g is constructed from NFA A in a way that allows each transition of A to replace any input subword by a word from L_2 of equal length. In addition, g has a new start state s_0 and an extra final state s_f, which are used to skip over any input prefix and/or suffix, outputting it without change. The production rules P' can be explained as follows. Rule (1) allows the gsm to output any prefix of its input without change. Rules (2) and (3) are responsible for the substitution of a random subword of its input by a string from L_2 of the same length. Rules (4) and (5) allow the gsm to output any suffix of its input without change. □

Corollary 2. The REG, CF and CS families of languages are closed under $\bowtie_b$ with regular languages.

Proposition 11. CF is not closed under $\bowtie_b$.

Proof:
Let $\Sigma = \{a, b, c, d, e\}$ and take two context-free languages

$$L_1 = \{a^n b^n c^m d^m \mid n, m \geq 1\} \text{ and } L_2 = \{e^n f^n \mid n \geq 1\}.$$

Let $L = (L_1 \bowtie_b L_2) \cap a^*e^*f^*d^* = \{a^n e^k f^n d^m \mid 2k \geq n+m, n+m \text{ even}, k \geq 1\}$. Observe that L is not a context-free language. □

Proposition 12. CS is closed under $\bowtie_b$.

Proof:
Let L_1, L_2 be two context-sensitive languages over Σ and let $\Sigma' = \{a' | a \in \Sigma\}$, $\Sigma'' = \{a'' \mid a \in \Sigma\}$. Let g be the same gsm that was defined in Proposition 9 and that transforms a nonempty subword of its input into its primed version. Let $h : \Sigma \to \Sigma''$ be the morphism defined as $h(a) = a'', a \in \Sigma$. Let $h' : \Sigma \cup \Sigma' \cup \Sigma'' \to \Sigma$ be the morphism defined as $h'(a) = a, h'(a') = \lambda, h'(a'') = a, a \in \Sigma$.

We claim that

$$L_1 \bowtie_b L_2 = h'[[g(L_1) \amalg h(L_2)] \cap R],$$

where R is as defined in Proposition 9.

Indeed, let $u = u_1 w u_2 \in L_1$ and $v \in L_2$, where $w = w_1 \dots w_n$, $v = v_1 \dots v_n$ and $u_i, w_i \in \Sigma$. Then

$$\begin{aligned}
&u_1 w_1' \dots w_n' u_2 \in g(L_1), \quad v_1'' \dots v_n'' \in h(L_2),\\
&u_1 w_1' v_1'' w_2' v_2'' \dots w_n' v_n'' u_2 \in [g(L_1) \amalg h(L_2)] \cap R,\\
&h'(u_1 w_1' v_1'' w_2' v_2'' \dots w_n' v_n'' u_2) = u_1 v_1 v_2 \dots v_n u_2 = u_1 v u_2 \in u \bowtie_b v.
\end{aligned}$$

Note that h' is a 2-linear erasing morphism with respect to the language it is applied to, as it erases at most half of each word. As CS is closed under linear erasing homomorphisms, intersection with regular languages, shuffle, it follows it is closed also under $\bowtie_b$. □

Proposition 13. CF, REG are closed under $\rhd_b$ with regular languages.

Proof:
Let L_2 be a regular language, $A = (S, \Sigma, s_0, F, P)$ be a finite automaton, $L(A) = L_2$. Let $\Sigma' = \{a' \mid a \in \Sigma\}$ and let $h : \Sigma' \cup \Sigma'' \to \Sigma$ be the homomorphism defined by $h(a') = \lambda, h(a'') = a$. Construct the gsm $g = (S, \Sigma, \Sigma \cup \Sigma', s_0, F, P')$, where

$$P' = \{pa \to a'q \mid pa \to q \in P\} \cup \{pb \to a''q \mid pa \to q \in P\}.$$

In the above productions, a and b are not necessarily distinct. If $u \in L_1, v \in L_2$, and $|u| = |v|$, then the first type of rules marks (with a single prime) subwords that are common between u and v. The second type of rules outputs marked (with a double prime) subwords of v which may or may not be common with u.

Observe that

$$L_1 \rhd_b L_2 = h(g(L_1) \cap [(\Sigma')^*(\Sigma'')^*(\Sigma')^*]).$$

Recall that REG, CF are closed under gsm mappings and other operations used. Hence REG, CF are also closed under $\rhd_b$ with regular languages. □

Proposition 14. The family of CF languages is not closed under $\rhd_b$.

Proof:
Consider the languages

$$L_1 = \{a^n b^{3n} \mid n \geq 1\}, \text{ and}$$
$$L_2 = \{a^n c^m d^m a^n \mid n, m \geq 1\}.$$

Then $(L_1 \rhd_b L_2) \cap c^+ d^+ a^+ = \{c^i d^i a^j \mid i + j \text{ is even and } 1 \leq j \leq i\}$, which is not a context-free language, and since CF is closed under intersection with regular languages, the statement of the proposition follows. □

Proposition 15. The family of CS languages is not closed under $\rhd_b$. In particular, there exists a regular language R and a CS language L such that $R \rhd_b L$ is not a CS language.

Proof:
The proof depends on a result from [16], which is Theorem 9.9 (p. 89). The statement of the theorem is as follows.

Let L be a recursively enumerable language over the alphabet Σ and let $a, b \notin \Sigma$. Then there is a CS language L_1 such that (i) L_1 consists of words of the form $a^i bP$ where $i \geq 0$ and $P \in L$, and (ii) for every $P \in L$, there is an $i \geq 0$ such that $a^i bP \in L_1$.

Let L be some language over alphabet Σ which is recursively enumerable, but not context-sensitive. Take L_1 to be the CS language in the statement of the above result, i.e.,

$$L_1 = \{a^i bu \mid i \geq 0, u \in L\}, \quad a, b \notin \Sigma.$$

Let $L_1' = L_1\$$, where $\$$ is a symbol not in $\Sigma \cup \{a, b\}$. L_1' is also a CS language. Consider the regular language

$$R = \{a^i \# w \# \mid i \geq 0, w \in \Sigma^*\}, \quad \# \notin \Sigma \cup \{a, b\},$$

where a, b are the special symbols mentioned above. Let $h : \Sigma \cup \{b, \$\} \to \Sigma$ be the homomorphism defined by

$$h(b) = \lambda,\ h(\$) = \lambda,\ h(c) = \ c \ for\ all\ c \in \Sigma.$$

Then

$$L = h((R \rhd_b L_1') \cap b\Sigma^*\$).$$

Notice that h is k-linear erasing with respect to the language it is applied to. This is because it only erases at most two letters from any word, so we could say it is 2-linear erasing. Since the family of CS languages is closed under linear erasing and intersection with regular languages, the statement of the proposition follows. □

The following two examples illustrate that for an operation $\diamondsuit \in \{\bowtie_b, \rhd_b\}$ there exists a regular language R and a context-free language L, such that $R \diamondsuit L$ is not regular.

Example 4.1. There exists a regular language R and a CF language L, such that $R \bowtie_b L$ is not regular. For instance, let R be the language c^* and L be the language $\{a^n b^n \mid n \geq 1\}$. Then $R \bowtie_b L$ is the language $c^* a^n b^n c^*$, which is not regular.

Example 4.2. There exists a regular language R and a context-free language L, such that $R \rhd_b L$ is not regular. For instance,

$$R = c\$(\{a\} \cup \{b\})^*\$d,$$
$$L = \{c\#a^n b^n \# d \mid n \geq 1\}.$$

Then $(R \rhd_b L) \cap (\#a^*b^*\#) = \#a^n b^n \#$, which is not regular.

To summarize this section, for $\diamondsuit_b \in \{\bowtie_b, \triangle_b, \rhd_b\}$ we have that REG and CF are closed under $\diamondsuit_b$ with REG, while CF and CS are not closed under $\diamondsuit_b$.

5. Language Equations

In this section we study language equations of the types $X \diamondsuit L = R$ and $L \diamondsuit Y = R$, where X and Y are the unknowns and $\diamondsuit$ is one of the block substitution operations. We answer questions about the existence of a solution to a given language equation.

We employ a result from [8] that uses the left and right inverse operations to solve language equations.

Theorem 2. Let $L, R \subseteq \Sigma^*$ be two languages and let $\diamondsuit$ be a binary word operation. If the equation $X \diamondsuit L = R$ (respectively $L \diamondsuit Y = R$) has a solution, then the language $X_{max} = (R^c \diamondsuit^l L)^c$ (respectively $Y_{max} = (L \diamondsuit^r R^c)^c$) is also a solution, namely one that includes all the other solutions to the equation.

Proposition 16. The problem "Does there exist a solution Y to the equation $L \bowtie_b Y = R$?" is decidable for regular languages L and R.

Proof:
For given regular languages L, R over an alphabet Σ define $R' = (L \rhd_b R^c)^c$. From the above theorem it follows that if there exists a solution $Y \subseteq \Sigma^*$ to the equation $L \bowtie_b Y = R$, then $L \bowtie_b R' = R$.

Moreover, the regular solution R' can be effectively constructed. The algorithm which decides the problem begins by constructing R' and then testing whether or not $L \bowtie_b R'$ equals R. □

Proposition 17. The problem "Does there exist a singleton solution $Y = \{w\}$ to the equation $L \bowtie_b Y = R$?" is decidable for regular languages L and R.

Proof:
Let L, R be nonempty regular languages over an alphabet Σ and let m be the length of the shortest word in R. If there exists a word w such that $L \bowtie_b \{w\} = R$, then it must satisfy the condition $lg(w) \leq m$. The algorithm for deciding the problem consists of checking whether or not $L \bowtie_b \{w\} = R$ for all words w with $lg(w) \leq m$. This can be decided because $L \bowtie_b \{w\}$ is a regular language (by Cor. 2) and set equality is decidable for regular languages. The answer of the algorithm is YES if such a word w is found and NO otherwise. □

Proposition 18. The problem " Does the equation $L \bowtie_b Y = R$ have a solution Y?" is undecidable for context-free L and regular R.

Proof:
Let Σ be an alphabet and let $\#$ be a letter which does not occur in Σ. There exists a regular language $R = \#\Sigma^* \cup \#(\Sigma^* \amalg \{\#\})$ such that the problem of the proposition is undecidable for a context-free language L.

Let $L' \subseteq \Sigma^*$ be a context-free language and consider the language $L = \#L'$. We claim that for all languages $Y' \subseteq \Sigma^*$ the equation:

$$\#L' \bowtie_b (Y'\#) = \#\Sigma^* \cup \#(\Sigma^* \amalg \{\#\})$$

holds if and only if $Y' = \{\lambda\}$ and $L' = \Sigma^*$. Indeed, suppose that $Y' = \{\lambda\}$ and $L' = \Sigma^*$. Then

$$\#L' \bowtie_b (Y'\#) = \#\Sigma^* \bowtie_b \{\#\} = \#\Sigma^* \cup \#(\Sigma^* \amalg \{\#\}) = R.$$

For the other direction now, suppose that $\#L' \bowtie_b (Y'\#) = R$, where

$$R = \#\Sigma^* \cup \#(\Sigma^* \amalg \{\#\}).$$

It must be that $Y' = \{\lambda\}$. Otherwise we could obtain a word in R that does not begin with $\#$, which is a contradiction. The fact that $Y' = \{\lambda\}$ implies that $L' = \Sigma^*$.

Hence the claim holds. If we could decide the problem of the proposition, we could also decide whether for a given context-free language L, there exists a solution Y to the equation $L \bowtie_b Y = \#\Sigma^* \cup \#(\Sigma^* \amalg \{\#\})$. According to the claim proved above, this would in turn imply that we could decide the problem "Is $L' = \Sigma^*$?", which is impossible for context-free languages. □

Noticing that in the above proof $Y = \{\lambda\}$ is a singleton language, the proof can be used to show that the problem "Does there exist a singleton solution $Y = \{w\}$ to the equation $L \bowtie_b Y = R$?" is undecidable for context-free languages L and regular languages R.

We now turn to examining the language equation $X \bowtie_b L = R$. We can obtain the following result, using the same method as in Proposition 16.

Proposition 19. The problem "Does there exist a solution X to the equation $X \bowtie_b L = R$?" is decidable for regular languages L and R.

Proof:
For given regular languages L, R over an alphabet Σ define $R' = (R^c \rhd_b L)^c$. It follows that if there exists a solution $X \subseteq \Sigma^*$ to the equation $X \bowtie_b L = R$, then $R' \bowtie_b L = R$.

Moreover, the regular solution R' can be effectively constructed. The algorithm which decides the problem begins by constructing R' and then testing whether or not $R' \bowtie_b L$ equals R. □

Proposition 20. The problem "Does there exist a singleton solution $X = \{w\}$ to the equation $X \bowtie_b L = R$?" is decidable for regular languages L and R.

Proof:
Let L, R be nonempty regular languages over an alphabet Σ. Observe that if such a solution $X = \{w\}$ exists, then all words of R must be of the same length, namely $|w|$, i.e. R is a block code. Hence if R is infinite or not a block code, then the answer to the problem is NO. Otherwise, if R is a block code

of length m, then the decision algorithm checks whether or not $X \bowtie_b L = R$ for all words w of length m. This can be decided because $X \bowtie_b L$ is a finite language and set equality is decidable for finite languages. The answer of the algorithm is YES if such a word w is found and NO otherwise. □

We now show that if L is a regular language, the existence of solution to the equation is undecidable.

Proposition 21. The problem "Does the equation $X \bowtie_b L = R$ have a solution X?" is undecidable for context-free L and regular R.

Proof:
Let Σ be an alphabet and let $\#$ be a letter which does not occur in Σ. There exists a regular language $R = \Sigma^*$ such that the problem of the proposition is undecidable for context-free languages L. We assume the contrary and show how to solve the problem "Is $L' - L'' = \emptyset$?" for context-free languages L' and L''. For given context-free L', L'', define:

$$L = \#(L' - L'') \cup \Sigma^*.$$

We claim that there exists a language X such that $X \bowtie_b L = R$ iff $L' - L'' = \emptyset$, where L and R are defined as above. Indeed, if $L' - L'' = \emptyset$, then $L = \Sigma^*$ and $X = \Sigma^*$ is a solution to $X \bowtie_b L = R$. On the other hand, suppose there exists X such that $X \bowtie_b L = R$. Then $L' - L'' = \emptyset$. Indeed, if $L' - L'' \neq \emptyset$, then take $u \in (L' - L'')$, implying $\#u \in L$. Take $w \in X$ such that $|w| \geq |\#u|$ (such w exists as X has to be infinite to satisfy $X \bowtie_b L = \Sigma^*$). Then $w \bowtie_b (\#u) \in R$, which contradicts the structure of R.

The problem "Is $L' - L'' = \emptyset$?" is undecidable for context-free L', L'', hence the problem of the proposition is undecidable. □

The operations $\triangle_b$ and $\triangleright'_b$ are right-inverses of each other and so are the operations $\triangleright_b$ and $\bowtie_b$. The family of regular languages is closed under all four of these operations.

Using similar reasons as in Proposition 16, we have the following results.

The problem "Does there exist a solution Y to the equation $L \triangle_b Y = R$ (resp. $L \triangleright_b Y = R$)?" is decidable for regular languages L and R. If the solution Y exists, then $R' = (L \triangleright'_b R^c)^c$ (resp.$R' = (L \bowtie'_b R^c)^c$) is a solution as well. Moreover, R' includes all other solutions of the equation.

Proposition 22. The problem "Does there exist a singleton solution $Y = \{w\}$ to the equation $L \triangle_b Y = R$?" is decidable for regular languages L and R.

Proof:
Similar to that of Proposition 17. □

Proposition 23. The problem "Does the equation $L \triangle_b Y = R$ have a solution Y?" is undecidable for context-free L and regular R.

Proof:
Let Σ be an alphabet and let $\#$ and $\$$ be letters which do not occur in Σ. There exists a regular language

$$R = \bigcup_{a,b \in \Sigma \cup \{\#,\$\}} ab\Sigma^*$$

such that the problem of the proposition is undecidable for context-free languages L. Let $L' \subseteq \Sigma^*$ be a context-free language and consider the language

$$L = \#\$L' \cup \#\Sigma^*\$.$$

We claim that for all languages $Y' \subseteq \Sigma^*$ the equation:

$$(\#\$L' \cup \#\Sigma^*\$) \ \triangle_b \ (\#Y'\$) = \bigcup_{a,b \in \Sigma \cup \{\#,\$\}} ab\Sigma^*$$

holds if and only if $Y' = \{\lambda\}$ and $L' = \Sigma^*$. Indeed, suppose that $Y' = \{\lambda\}$ and $L' = \Sigma^*$. Then

$$(\#\$L' \cup \#\Sigma^*\$) \ \triangle_b \ (\#Y'\$) = (\#\$\Sigma^* \cup \#\Sigma^*\$) \ \triangle_b \ (\#\$) = \bigcup_{a,b \in \Sigma \cup \{\#,\$\}} ab\Sigma^* = R.$$

Suppose now that $(\#\$L' \cup \#\Sigma^*\$) \ \triangle_b \ (\#Y'\$) = R$, where R is as defined above. It must be that $Y' = \{\lambda\}$. Otherwise, we could obtain a word in R that is of the form $\#w\$$, where $w \neq \lambda$, which is a contradiction. Indeed, if $w \neq \lambda$ is in Y', then since we have $\#w\$ \in L$, we obtain $\#w\$ \in R$.

The fact that $Y' = \{\lambda\}$ implies that $L' = \Sigma^*$.

Hence the claim holds. If we could decide the problem of the proposition, we could also decide whether for a given context-free language L, there exists a solution Y to the equation

$$L \triangle_b Y = \bigcup_{a,b \in \Sigma \cup \{\#,\$\}} ab\Sigma^*.$$

According to the claim proved above, this would in turn imply that we could decide the problem "Is $L' = \Sigma^*$?", which is impossible for context-free languages. □

Noticing that in the above proof $Y = \{\lambda\}$ is a singleton language, the proof can be used to show that the problem "Does there exist a singleton solution $Y = \{w\}$ to the equation $L \bowtie_b Y = R$?" is undecidable for context-free languages L and regular languages R.

We now turn to examining the language equation $X \triangle_b L = R$. We can obtain the following results, using the same methods as in Propositions 16 and 20 respectively.

Proposition 24. The problem "Does there exist a solution X to the equation $X \triangle_b L = R$?" is decidable for regular languages L and R.

Proposition 25. The problem "Does there exist a singleton solution $X = \{w\}$ to the equation $X \triangle_b L = R$?" is decidable for regular languages L and R.

Currently it is not known whether the problem "Does the equation $X \triangle_b L = R$ have a solution X?" is decidable for context-free L and regular R. (Probably undecidable.)

Acknowledgements

The authors would like to thank Mike Domaratzki for helpful suggestions regarding Section 5. This research was partially supported by the Canada Research Chair Grant and NSERC Discovery Grant R2824A01.

References

[1] J. Berstel, D. Perrin: *Theory of Codes*. Academic Press, Orlando, 1985.

[2] J.H. Conway: *Regular Algebra and Finite Machines*. Chapman and Hall, London, 1971.

[3] M. Crochemore, C. Hancart: Automata for matching patterns. In[15], vol. II, 399–462.

[4] T. Head, A. Weber: Deciding code related properties by means of finite transducers. In *Sequences II, Methods in Communication, Security, and Computer Science* (R. Capocelli, A. de Santis, U. Vaccaro, eds.), Springer-Verlag, Berlin, 1993, 260–272.

[5] H. Jürgensen, S. Konstantinidis: Codes. In [15], vol. I, 511–607.

[6] H. Jürgensen, K. Salomaa, S. Yu: Transducers and the decidability of independence in free monoids. *Theoretical Computer Science*, 134 (1994), 107–117.

[7] L. Kari: *On Insertion and Deletion in formal languages*. PhD thesis, University of Turku, Finland, 1991.

[8] L. Kari: On language equations with invertible operations. *Theoretical Computer Science*, 132 (1994), 129–150.

[9] L. Kari, S. Konstantinidis: Language equations, maximality and error-detection. Elsevier Science, 2004.

[10] L. Kari, G. Thierrin: K-catenation and applications: k-prefix codes. *Journal of Information and Optimization Sciences*, 16 (1995), 263–276.

[11] L. Kari, G. Thierrin: Maximal and minimal solutions to language equations. *Journal of Computer and System Sciences*, 53 (1996), 487–496.

[12] S. Konstantinidis: Relationships between different error-correcting capabilities of a code. *IEEE Trans. Inform. Theory*, 47 (2001), 2065–2069.

[13] S. Konstantinidis: Transducers and the properties of error-detection, error-correction, and finite-delay decodability. *J. Universal Comp. Science*, 8 (2002), 278–291.

[14] S. Konstantinidis, A. O'Hearn: Error-detecting properties of languages. *Theoretical Computer Science*, 276 (2002), 355–375.

[15] G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages*. Springer-Verlag, Berlin, 1997.

[16] A. Salomaa: *Formal Languages*. Academic Press, New York, 1973.

[17] H.J. Shyr: *Free monoids and languages*. Institute of Applied Mathematics, National Chung-Hsing University, Taichung, Taiwan, 1991.

[18] S. Yu: Regular Languages. In [15], vol. I, 41–110.