International Journal of Foundations of Computer Science
Vol. 22, No. 7 (2011) 1655–1668

DOI: 10.1142/S0129054111008945

World Scientific
www.worldscientific.com

SCHEMA FOR PARALLEL INSERTION AND DELETION: REVISITED

LILA KARI* and SHINNOSUKE SEKI†

Department of Computer Science, The University of Western Ontario
London, Ontario, N6A 5B7, Canada
**lila@csd.uwo.ca*
†*sseki@csd.uwo.ca*

Received 16 November 2010
Accepted 28 February 2011
Communicated by Sheng Yu

A general framework of parallel insertion and deletion based on p-schemata is proposed. A p-schema is a set of tuples of words. When being used for parallel insertion of a language into a word, an element of p-schema specifies how to split the word into factors between which the language to be inserted. As its inverse operation, parallel deletion based on the p-schema is defined. Several algebraic properties of these operations such as closure properties of language classes under them are proved. The main contribution of this paper is to propose algorithms to solve language equations involving p-schema-based insertions or deletions based on syntactic congruence. These algorithms not only decide whether a given equation has a solution but construct the set of all of its maximal or minimal solutions. The algorithms are extensible so as to solve some multiple-variables equations and inequalities. Related undecidability results are also presented.

Keywords: Contextual parallel insertion and deletion; p-schema; syntactic congruence; algorithms to solve multiple-variables language equations; language inequalities.

1. Introduction

Contextual insertion and deletion are our central interest in this paper. The fact that one can actually implement these operations in the laboratory [5] as sole primitives to model DNA computation gives practical significance to the research on these operations. They are also of theoretical interest; indeed, they have been proved Turing-universal [14]. The first aim of this paper is to propose a method to parallelize contextual insertion and deletion. For words x and y, (x, y)-contextual insertion of a language L into a word w [14] results in the language $\{w_1xvyw_2 \mid w_1, w_2$ are words such that $w = w_1xyw_2$ and $v \in L\}$. In other words, w is cut into two pieces so that the first piece ends with x and the second piece begins with y, and between them L is inserted. This operation suggests that for any

†Corresponding author.

positive integer n, an n-tuple $(w_1, w_2, \ldots, w_n)$ of words may implement insertion of $n-1$ instances of L into *the* word $w = w_1 w_2 \cdots w_n$ to generate the language $w_1 L w_2 L \cdots L w_{n-1} L w_n$. More generally, we have it that a *set* of tuples of words implements a parallel insertion so that we call such a set of tuples a *parallel operation schema* or *p-schema* for short. We call the parallel insertion thus implemented the *insertion based on the p-schema*. The use of p-schema is not used exclusively to parallelize contextual insertion. Parallel deletion of L from a word w based on a given n-tuple $(u_1, u_2, \ldots, u_n)$ deletes $n-1$ non-overlapping elements of L from w so as to leave this n-tuple, and concatenate them to generate the word $u = u_1 u_2 \cdots u_n$. That is to say, if $w \in u_1 L u_2 L \cdots L u_n$, then this deletion results in the word u, and otherwise it results in nothing. Indeed, various known sequential as well as parallel operations are special instances of insertion and deletion based on p-schemata (see Section 3).

Our main interest lies in the language equations which involve p-schema-based insertion and deletion. Solving equations of the form $X_1 \diamond X_2 = X_3$ is a core research object in formal language theory (see, e.g., [1, 3, 4, 7, 10, 11, 14]), where $\diamond$ is a binary operation on languages, some of X_1, X_2, X_3 are fixed languages (denoted by L, R with subscripts in this paper), and the others are unknowns (denoted by X, Y, Z). In this paper, we focus on such language equations with $\diamond$ being insertion (denoted by $\leftarrowtail_F$) or deletion (denoted by $\rightarrowtail_F$) based on a p-schema F. According to the definitions of p-schema-based insertion and deletion in Section 3, language equations $X \leftarrowtail_F L_2 = L_3$, $L_1 \leftarrowtail_X L_2 = L_3$, and their deletion variants are guaranteed to have a unique maximum solution as long as they have a solution, while neither $L_1 \leftarrowtail_F X = L_3$ nor $L_1 \rightarrowtail_F X = L_3$ possesses this property, and as such, a classical well-known technique proposed by Kari [11] is not applicable for solving these. Based on syntactic congruence, we will design algorithms to solve these equations, which lie at the center of our contributions in this paper. Our algorithms work not only as a procedure to decide the existence of solutions to a given equation, but as a procedure to construct all of its maximal solutions (Theorem 13 and Corollary 17). Moreover, combining these algorithms with the algorithms to solve the equations of the first or second form enables us to solve two-variables equations of the form $X \leftarrowtail_F Y = R_3$ (Theorem 19), $R_1 \leftarrowtail_X Y = R_3$ (Theorem 20), and $R_1 \rightarrowtail_X Y = R_3$ (Theorem 21) for regular languages R_1, R_2, R_3 and a regular p-schema F (a natural way to classify sets of tuples of words in the Chomsky-hierarchy manner is mentioned at the beginning of Section 4). The proposed algorithms have further extensibility so that a slight modification makes possible for them to solve inequality (set inclusion) variants of the above-mentioned equations. One of its applications of interest is that for a given regular language R, it can output the set of all maximal languages L satisfying $L^* \subseteq R$.

This is a journal version of our paper presented in DLT 2010 [13], but due to space limitation, this paper omits proofs and some results which appeared in [13], and hence, works rather as its complementary material.

2. Preliminaries

Let Σ be a finite alphabet, and let Σ^* denote the set of words over Σ, which includes the *empty word* λ. Let $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$. For a word $w \in \Sigma^*$, its length is denoted by $|w|$. A word u is called a *factor* (*prefix*, *suffix*) of a word w if $w = xuy$ (resp. $w = uy$, $w = xu$) for some words $x, y \in \Sigma^*$. Let us denote the set of all prefixes (suffixes) of w by $\mathrm{Pref}(w)$ (resp. $\mathrm{Suf}(w)$). For a language $L \subseteq \Sigma^*$, its complement is denoted by L^c, i.e., $L^c = \Sigma^* \setminus L$. For an integer $n \geq 0$, Σ^n, $\Sigma^{\leq n}$, and $\Sigma^{\geq n}$ denote the respective sets of all words of length *exactly* n, of length *at most* n, and of length *at least* n. As customary in formal language theory, a singleton language $\{w\}$ is quite often identified with its element w.

Regular languages are languages accepted by a finite automaton. We describe a (non-deterministic) finite automaton (NFA) by a 5-tuple $(Q, \Sigma, \delta, s, F)$, where Q is a finite set of states including the start state s and a set of final states F, and $\delta : Q \times \Sigma \to Q$ is a transition *relation*. An NFA is said to be *deterministic* if its transition relation is actually a function. The deterministic property of a machine is stated explicitly by denoting with the capital D. These hold for other acceptors. By REG, LIN, DCFL, and CFL, we denote the respective classes of regular, linear, deterministic context-free, and context-free languages.

A characterization of languages can be given in terms of *syntactic semigroup*. For a language $L \subseteq \Sigma^*$, there exists a maximal congruence $\equiv_L$ which saturates L (i.e., L is a union of equivalence classes in $\Sigma^*/\equiv_L$). This is called the *syntactic congruence* of L, which is defined as follows: for $u, v \in \Sigma^*$,

$$u \equiv_L v \iff \text{for any } x, y \in \Sigma^*,\ xuy \in L \text{ if and only if } xvy \in L.$$

For a word $w \in \Sigma^*$, $[w]_{\equiv_L} = \{u \in \Sigma^* \mid w \equiv_L u\}$ is called an *equivalence class* with w as its representative. The number of equivalence classes is called the *index* of $\equiv_L$.

Theorem 1 ([16]) *A language L is regular if and only if the index of $\equiv_L$ is finite.*

For technical reasons, we define a function called *saturator with respect to a language* L_1 as a function σ_{L_1} from a word w into the equivalence class $[w]_{\equiv_{L_1}}$. It is naturally extended for a language L as $\sigma_{L_1}(L) = \bigcup_{w \in L} [w]_{\equiv_{L_1}}$.

Theorem 2. *For a regular language R, each equivalence class in $\Sigma^*/\equiv_R$ is regular.*

3. Schema for Parallel Insertion and Deletion

Imagine that we will insert a language L into a word u in parallel. Let $\mathfrak{F} = \bigcup_{n \geq 1} \underbrace{\Sigma^* \times \Sigma^* \times \cdots \times \Sigma^*}_{n \text{ times}}$. As abstracted before, a subset F of $\mathfrak{F}$ can be used to control the parallel insertion of a language L in a sense that if $(u_1, u_2, \ldots, u_n) \in F$, then the word $u = u_1 u_2 \cdots u_n$ is split in the manner dictated by the n-tuple in F, and L is inserted between u_i and u_{i+1} for all $1 \leq i < n$ to generate the language

$u_1Lu_2L\cdots u_{n-1}Lu_n$. We will see that the set can also be used to implement a parallel deletion. For this intended end-usage, we call a subset of $\mathfrak{F}$ a *parallel operation schema*, or shortly *p-schema*, *over* Σ.

A p-schema F enables us to define the *insertion* $\leftharpoonup_F$ as: for a word $u \in \Sigma^*$ and a language $L \subseteq \Sigma^*$,

$$u \leftharpoonup_F L = \bigcup_{\substack{n\geq 1, u=u_1\cdots u_n,\\ (u_1,\ldots,u_n)\in F}} u_1Lu_2L\cdots u_{n-1}Lu_n.$$

Note that an n-tuple in F parallel-inserts $n-1$ words from L into u. Similarly, we define the *deletion* $\rightharpoonup_G$ *based on a p-schema* G as: for a word $w \in \Sigma^*$ and a language $L \subseteq \Sigma^*$,

$$w \rightharpoonup_G L = \{u_1\cdots u_n \mid n \geq 1, (u_1,\ldots,u_n) \in G, w \in u_1Lu_2L\cdots u_{n-1}Lu_n\}.$$

These operations are extended in a natural way for a language $L_1 \subseteq \Sigma^*$ as $L_1 \leftharpoonup_F L = \bigcup_{u\in L_1} u \leftharpoonup_F L$ and $L_1 \rightharpoonup_G L = \bigcup_{u\in L_1} u \rightharpoonup_G L$.

The following well-known operations are particular cases of p-schema-based insertion and deletion: catenation and right quotient ($F_{\mathrm{cat}} = \Sigma^* \times \lambda$), reverse catenation and left quotient ($F_{\mathrm{rcat}} = \lambda \times \Sigma^*$), sequential insertion and deletion ($F_{\mathrm{sins}} = \Sigma^* \times \Sigma^*$), and parallel insertion and deletion ($F_{\mathrm{pins}} = \bigcup_{n\geq 0}(\lambda \times \underbrace{\Sigma \times \cdots \times \Sigma}_{n \text{ times}} \times \lambda)$).

Example 3. *Parallel insertion (deletion) of exactly n words, at most n words, or arbitrary number of words are instances of insertion (resp. deletion) based on:*

$$F_{\mathrm{pins}(n)} = \underbrace{\Sigma^* \times \cdots \times \Sigma^*}_{n+1 \text{ times}}, \qquad F_{\mathrm{pins}(\leq n)} = \bigcup_{i=0}^{n} F_{\mathrm{pins}(i)}, \qquad F_* = \bigcup_{i=0}^{\infty} F_{\mathrm{pins}(i)}.$$

Using F_**, even the unary operation Kleene-star is implemented as* $L^* = \{\lambda\} \leftharpoonup_{F_*} L$.

All operations mentioned so far are "syntactic" in the sense that they do not rely on the context of their first operand to determine where to insert their second operand, and hence, we can say that the corresponding p-schemata are "syntactic." On the other hand, p-schemata have a potential to endow insertion (deletion) with *semantic control* on where to insert (resp. delete). *C-contextual (sequential) insertion* [14] is a good example, where $C \subseteq \Sigma^* \times \Sigma^*$. This is an insertion based on the p-schema $F_{\mathrm{csins}}(C) = \bigcup_{(x,y)\in C} \Sigma^* x \times y\Sigma^*$. The following p-schema naturally parallelizes this operation as *C-contextual parallel insertion*:

$$F_{\mathrm{cpins}}(C) = \{(u_1,\ldots,u_n) \mid n \geq 1, \forall 1 \leq i < n, \big(\mathrm{Suf}(u_i) \times \mathrm{Pref}(u_{i+1})\big) \cap C \neq \emptyset\}.$$

It may be worth noting that our framework and *I-shuffle* proposed by Domaratzki, Rozenberg, and Salomaa [6] are incomparable in the above sense. Indeed, only I-shuffle can specify contexts not only on the left operand but also on the right operand, while p-schemata allow operations to insert or delete more than one copies of right operand. Thus, the insertion based on a p-schema which is a subset of $F_{\mathrm{pins}(\leq 1)}$ is a special instance of I-shuffle.

4. Hierarchy of p-Schemata and Closure Properties

Besides the class of syntactic p-schemata, one can come up with various classifications of p-schemata. One natural way to classify them is according to how strong computational power is required to recognize them. Using a special symbol $\# \notin \Sigma$ as "linker", we can convert a tuple of words over Σ into a language over $\Sigma \cup \{\#\}$. A mapping $\psi : \mathfrak{F} \to (\Sigma \cup \{\#\})^*$ defined as $\psi((u_1, u_2, \ldots, u_{n-1}, u_n)) = u_1\#u_2\#\cdots\#u_{n-1}\#u_n$ achieves this task. Extending ψ to p-schemata brings us the notion of *p-schema language* $\psi(F)$ for a p-schema F, that is, $\psi(F) = \{\psi(f) \mid f \in F\}$. A one-to-one relationship between p-schemata and p-schema languages is clear, and as such, it is natural to say that a p-schema F is *in* a language class $\mathcal{L}$ if $\psi(F) \in \mathcal{L}$. For instance, a p-schema F is *regular* if $\psi(F)$ is regular: that is to say, being encoded by ψ, a finite automaton recognizes F.

In this section, we investigate closure properties of classes of finite-state machines augmented with one-reversal counters [2, 8, 9] under the p-schema-based insertions and deletions. A *counter* is a pushdown stack whose alphabet is unary so that one can check whether a counter is zero or non-zero, but cannot tell the exact count stored in the counter. A *one-reversal(-bounded)* counter is a counter which can change from its non-decrementing mode to non-incrementing mode only once during any computation.

By augmenting an NFA with k one-reversal counters, a machine called *one-reversal k-counter machine* can be defined. A one-reversal k-counter machine is formally represented by a 6-tuple $M = (k, Q, \Sigma, \delta, s, F)$, where Q, Σ, s, F are defined in the same way as done for NFA, while δ is a relation from $Q \times \Sigma \times \{0, 1\}^k$ into $Q \times \{-1, 0, +1\}^k$. A configuration of M is given by a $(k+2)$-tuple $(q, w, c_1, c_2, \ldots, c_k)$ denoting the fact that M is in the state q, w is the pending input, and $c_1, \ldots, c_k$ are the counts stored in the k counters. Then we can define a relation $\Rightarrow$ among configurations as: $(q, aw, c_1, \ldots, c_k) \Rightarrow (p, w, c_1 + d_1, \ldots, c_k + d_k)$ if $\delta(q, a, \zeta(c_1), \ldots, \zeta(c_k))$ contains $(p, d_1, \ldots, d_k)$, where $\zeta(c_i) = 0$ if $c_i = 0$ and $\zeta(c_i) = 1$ if $c_i > 0$. We assume that δ is defined carefully so as to prevent the counters from storing a negative count. Let NCM(k) be the class of one-reversal k-counter machines, and NCM be the union of such classes over all k's.

The class of machines obtained by further augmenting a one-reversal k-counter machine with one pushdown stack is denoted by NPCM(k). NPCM is the union of NPCM(k) over all k's. Any notation obtained by replacing N with D denote the corresponding deterministic subclass. It is especially worth of note that NCM(0) = REG, DPCM(0) = DCFL, and NPCM(0) = CFL.

Now we start our study on the closure properties of non-deterministic classes NCM and NPCM under p-schema-based insertions and deletions. First of all, we prove that REG is closed under insertion and deletion based on a regular p-schema as a corollary of the following stronger result.

Proposition 4. *For $L_1 \in$ NCM(k_1), a regular language R_2, and an* NCM(k_ψ) *p-schema F, both $L_1 \leftharpoonup_F R_2$ and $L_1 \rightharpoonup_F R_2$ are in* NCM($k_1 + k_\psi$).

Proof. Let A_2 be a finite automaton for R_2, and M_1, M_ψ be NCMs with k_1, k_ψ counters for $L_1, \psi(F)$, respectively. We describe two respective ways of constructing an NCM M with $k_1 + k_\psi$ counters which accepts $L_1 \leftharpoonup_F R_2$ and $L_1 \rightharpoonup_F R_2$.

Let us begin with the p-schema-based insertion. M expects its input to be of the form $u_1x_1u_2x_2\cdots x_{n-1}u_n$ for some integer $n \geq 1$, $u_1u_2\cdots u_n \in L_1$, $x_1, x_2, \ldots, x_{n-1} \in R_2$, and $u_1\#u_2\#\cdots\#u_n \in \psi(F)$. M simulates M_1 and M_ψ simultaneously. Guessing non-deterministically that the prefix $u_1x_1\cdots x_{i-1}u_i$ has been read, M pauses the simulation of both M_1 and M_ψ and instead activates the simulation of A_2 on x_i after having M_ψ make a #-transition. When A_2 is in one of its accepting states, M non-deterministically resumes the simulation of M_1 and M_ψ on the suffix $u_{i+1}x_{i+1}\cdots x_{n-1}u_n$ of the input. The simulation of A_2 is initialized every time it is invoked.

For the p-schema-based deletion, M assumes that its input can be split into $u_1, u_2, \ldots, u_n$. This split is no more than a non-deterministic guess, and M takes all of such splits into consideration. For each i, M first simulates M_1 and M_ψ on u_i. Then M pauses M_ψ after having it make a #-transition, and without moving its input head, M guesses symbol-by-symbol a string x_i and on this string continues the simulation of M_1 as well as invokes and runs A_2. When A_2 accepts the guessed x_i, M ends the simulation of A_2, and proceeds to the process for u_{i+1}. □

Corollary 5. *For regular languages R_1, R_2 and a regular p-schema F, both $R_1 \leftharpoonup_F R_2$ and $R_1 \rightharpoonup_F R_2$ are regular and effectively constructible.*

In the problem setting considered in Proposition 4, if the recognition of L_2 requires a counter (i.e., $L_2 \in \mathrm{NCM}(k)$ for some $k \geq 1$), then it might be the case that $L_1 \leftharpoonup_F L_2$ or $L_1 \rightharpoonup_F L_2$ is not in NCM. This is because we might have to call the one-reversal counter machine for L_2 arbitrary many times, and each time at least 1 one-reversal counter is "consumed." Therefore, if we know beforehand at most how many times L_2 to be inserted (deleted), then we can resource the machine with sufficient amount of counters.

Corollary 6. *Let $L_1 \in \mathrm{NCM}(k_1)$, $L_2 \in \mathrm{NCM}(k_2)$, and F be a p-schema in $\mathrm{NCM}(k_\psi)$. If there exists an integer $n \geq 0$ such that $F \subseteq F_{\mathrm{pins}(\leq n)}$, then both $L_1 \leftharpoonup_F L_2$ and $L_1 \rightharpoonup_F L_2$ are in $\mathrm{NCM}(k_1 + nk_2 + k_\psi)$.*

Let us enlarge our interest onto the closure properties of NPCM. Even in the cases when we extend some classes which L_1, L_2, or F belongs to NPCM, the proof idea of Proposition 4 works. We present two of such cases here.

Proposition 7. *For $L_1 \in \mathrm{NPCM}(k_1)$, $L_2 \in \mathrm{CFL}$, and an $\mathrm{NCM}(k_\psi)$ p-schema F, $L_1 \leftharpoonup_F L_2$ is in $\mathrm{NPCM}(k_1 + k_\psi)$.*

Proof. Let M_1 be an $\mathrm{NPCM}(k_1)$ for L_1, M_2 be a pushdown automaton for L_2, and M_ψ be an $\mathrm{NCM}(k_\psi)$ for $\psi(F)$. We can construct an NPCM M accepting $L_1 \leftharpoonup_F L_2$ as done in the proof of Proposition 4. Indeed, it suffices to describe the strategy of

how M_1 and M_2 share only one pushdown store in M. When pausing the simulation of M_1 in order to activate M_2, M marks the top of the stack by some special symbol so that M can resume the simulation of M_1 with the same stack after the simulation of M_2. For this strategy to work, the stack alphabet must be non-unary. □

Analogous to Corollary 5, the case of $k_1 = k_\psi = 0$ in Proposition 7 is of interest; CFL is closed under insertion based on a regular p-schema. The next result should be clear from the proof of Proposition 7.

Proposition 8. *For* $L_1 \in \mathrm{NCM}(k_1)$, $L_2 \in \mathrm{CFL}$, *and an* $\mathrm{NPCM}(k_\psi)$ *p-schema* F, $L_1 \leftarrow\!\!\prec_F L_2$ *is in* $\mathrm{NPCM}(k_1 + k_\psi)$.

Note that our construction of M (for insertion) described in Proposition 4 cannot let both L_1 and F require a pushdown stack because M needs to run machines for these languages at the same time. In addition, that of M for deletion needs to run also machines for L_1 and L_2 simultaneously. Thus, the deletion variant of Proposition 8 holds, while we cannot say that that of Proposition 7 holds.

To argue the closure properties of non-deterministic counter machine classes further would carry us too far away from the purpose of this paper; algorithms to solve language equations or to decide whether a given equation has a solution. This decision task turns out to be undecidable very likely once some language involved is allowed to be NCM(1) because it is undecidable for an arbitrary $L \in \mathrm{NCM}(1)$ whether $L = \Sigma^*$ [9]. By contrast, the deterministic classes DCM and DPCM possess the following decidability property as proved by Ibarra as Corollary 5.4 in [9].

Theorem 9 ([9]) *For arbitrary* $L_1 \in \mathrm{DCM}(k_1)$ *and* $L_2 \in \mathrm{DPCM}(k_2)$ *with* $k_1, k_2 \geq 0$, *it is decidable whether* $L_1 = L_2$.

5. Language Equations with *p*-Schema-Based Insertions and Deletions

Having defined p-schema-based insertion and deletion properly and proved algebraic properties of these operations, now we commence the investigation on language equations with these operations. The last decades saw the intensive investigations on language equations with special instances of p-schema-based insertions and deletions such as catenation and quotient, sequential insertion and deletion as well as incomparable operations like shuffle [3, 7, 10, 11, 15]. A method to solve language equations which are guaranteed to have the maximum solution (if it has any) has been established by Kari [11] based on the notion of inverseness of operations, and it can be extended to solve $X \leftarrow\!\!\prec_F L_2 = L_3$, $L_1 \leftarrow\!\!\prec_X L_2 = L_3$, and their deletion variants [13]. In this section, we propose algorithms to solve $L_1 \leftarrow\!\!\prec_F X = L_3$ and its deletion variant for the case of L_1, L_3, F being regular.

Let us begin our investigation with exemplifying that such equations can have multiple maximal solutions as follows:

Example 10. *Let* $L_{\mathrm{even}} = \{a^{2n} \mid n \geq 0\}$, $L_{\mathrm{odd}} = \{a^{2m+1} \mid m \geq 0\}$, *and* $F = F_{\mathrm{pins}(2)} \cup F_{\mathrm{pins}(0)}$. *Both* L_{even} *and* L_{odd} *are (maximal) solutions to* $L_{\mathrm{even}} \leftharpoonup_F X = L_{\mathrm{even}}$ *and* $L_{\mathrm{even}} \rightharpoonup_F X = L_{\mathrm{even}}$. *On the other hand,* $L_{\mathrm{even}} \leftharpoonup_F (L_{\mathrm{even}} \cup L_{\mathrm{odd}})$ *and* $L_{\mathrm{even}} \rightharpoonup_F (L_{\mathrm{even}} \cup L_{\mathrm{odd}})$ *can generate the elements of* L_{odd}.

5.1. *Solving* $L_1 \leftharpoonup_F X = L_3$

Let us propose one approach to solve the existence of right operand. The idea originates from Conway (Chapter 6 of [3]) to solve $f(\Sigma \cup \{X_1, X_2, \ldots\}) \subseteq R$, where f is a regular function over Σ and variables $X_1, X_2, \ldots$, and R is a regular language, based on syntactic congruence. Here the idea is briefly explained in terms of p-schema-based insertions and deletions in order to take a step into more general cases than the case when both L_1 and F are regular.

Lemma 11. *Let* $L, L_1 \subseteq \Sigma^*$ *be languages and* F *be a p-schema. Then for any word* $w \in \Sigma^*$ *and language* $L_2 \subseteq \Sigma^*$, $(L_1 \leftharpoonup_F (L_2 \cup \{w\})) \cap L \neq \emptyset$ *if and only if* $(L_1 \leftharpoonup_F (L_2 \cup [w]_{\equiv_L})) \cap L \neq \emptyset$, *where* $\equiv_L$ *is the syntactic congruence of* L.

Proof. Assume the non-emptiness of $(L_1 \leftharpoonup_F (L_2 \cup [w]_{\equiv_L})) \cap L$. This means that there exist $n \geq 1$, $u_1, \cdots, u_n \in \Sigma^*$, and $x_1, \cdots, x_{n-1} \in L_2 \cup [w]_{\equiv_L}$ such that $u_1 \cdots u_n \in L_1$ and $u_1 x_1 u_2 x_2 \cdots u_{n-1} x_{n-1} u_n \in (L_1 \leftharpoonup_F (L_2 \cup [w]_{\equiv_L})) \cap L$. Consider a word $u_1 x_1' u_2 x_2' \cdots u_{n-1} x_{n-1}' u_n$, where $x_i' = w$ if $x_i \in [w]_{\equiv_L}$ or $x_i' = x_i$ otherwise. This word is in $L_1 \leftharpoonup_F (L_2 \cup w)$, and by the definition of syntactic congruence $\equiv_L$, this word is also in L. Thus, $(L_1 \leftharpoonup_F (L_2 \cup w)) \cap L$ is not empty. □

Imagine that we are given a "safe" partial solution X in a sense that $L_1 \leftharpoonup_F X \subseteq L_3$. Substituting L_3^c into L in Lemma 11 gives us one implication of importance to solve $L_1 \leftharpoonup_F X = L_3$. That is to say, if $X \cup \{w\}$ is guaranteed to be safe in this sense, $X \cup [w]_{\equiv_{L_3}}$ is also safe (recall that $\equiv_{L_3}$ is equivalent to $\equiv_{L_3^c}$). One interpretation of this is that we can handle equivalence classes as an "atomic unit" as long as maximal solutions are concerned. Hence, Lemma 11 encourages us to introduce the notion of syntactic solution. Given a language L, we say that a solution to a one-variable language equation is *syntactic with respect to* L if it is a union of equivalence classes in $\Sigma^*/\equiv_L$. The previous lemma implies that if L_2 is its solution, then $\sigma_{L_3}(L_2)$ is its syntactic solution.

Proposition 12. *For languages* L_1, L_3 *and a p-schema* F, *the equation* $L_1 \leftharpoonup_F X = L_3$ *has a solution if and only if it has a syntactic solution with respect to* L_3.

As a result, the problem of deciding whether $L_1 \leftharpoonup_F X = L_3$ has a solution is reduced to the problem of deciding whether this equation has a syntactic solution. If L_3 is regular, then the number of all of its syntactic solutions is finite (Theorem 1), and the syntactic solution is regular (Theorem 2). Let us present a pseudocode of the algorithm to solve the equation in this case. Let $ß = \bigcup_{L \subseteq \Sigma^*} \sigma_{R_3}(L)$, the set of all candidates of syntactic solutions.

Algorithm to solve $L_1 \leftarrow\!\prec_F X = R_3$

(1) Construct ß and order its elements in some way (let us denote the i-th element of ß by ß$[i]$).
(2) for each $1 \leq i \leq |ß|$, test whether $L_1 \leftarrow\!\prec_F ß[i]$ is equal to R_3.

Due to the algorithm above, Theorem 9, and Corollary 5, it is decidable whether $R_1 \leftarrow\!\prec_F X = R_3$ has a solution or not for regular languages R_1, R_3 and a regular p-schema F. However, this algorithm provides us with a stronger result than the decidability. Note that all maximal solutions of $L_1 \leftarrow\!\prec X = L_3$ are syntactic because σ_{L_3} saturates a non-syntactic solution, and a properly-bigger syntactic solution results. As a result, if L_3 is regular, then the equation $L_1 \leftarrow\!\prec X = L_3$ has at most a finite number of maximal solutions.

Theorem 13. *For regular languages R_1, R_3 and a regular p-schema F, the set of all maximal solutions to $R_1 \leftarrow\!\prec_F X = R_3$ is effectively constructible.*

The regularity of L_3 is necessary for the algorithm to solve $L_1 \leftarrow\!\prec_F X = L_3$, whereas such a condition is not imposed on L_1 or on F. Since the equality test is decidable between DPCM and REG (Theorem 9), under conditions that substituting any union of equivalence classes in $\Sigma^*/\equiv_{L_3}$ into $L_1 \leftarrow\!\prec_F X$ yields a DPCM, the existence of a solution to $L_1 \leftarrow\!\prec_F X = L_3$ remains decidable. Finding such conditions, however, seems to be hard as suggested in Proposition 2 of [13].

5.2. *Solving $L_1 \rightarrowtail_F X = L_3$*

A similar approach based on the syntactic congruence works for the equations of the form $L_1 \rightarrowtail_F X = L_3$.

Lemma 14. *Let $L_1 \subseteq \Sigma^*$ be a language and F be a p-schema. Then for any word $w \in \Sigma^*$ and language $L_2 \subseteq \Sigma^*$, $L_1 \rightarrowtail_F (L_2 \cup \{w\}) = L_1 \rightarrowtail_F (L_2 \cup [w]_{\equiv_{L_1}})$.*

Proof. It suffices to prove that $L_1 \rightarrowtail_F (\{w\} \cup L_2) \supseteq L_1 \rightarrowtail_F ([w]_{\equiv_{L_1}} \cup L_2)$ holds. Let $u \in L_1 \rightarrowtail_F ([w]_{\equiv_{L_1}} \cup L_2)$. This means that there exist $v \in L_1$, $n \geq 0$, $(u_1, u_2, \ldots, u_{n+1}) \in F$, and $x_1, \ldots, x_n \in [w]_{\equiv_{L_1}} \cup L_2$ such that $u = u_1 u_2 \cdots u_{n+1}$ and $v = u_1 x_1 u_2 x_2 \cdots u_n x_n u_{n+1}$. Now on v if $x_i \in [w]_{\equiv_{L_1}}$, then we replace x_i with w, and this process converts v into a word v'. Note that this replacement process guarantees that $v' \in L_1$ because the replaced factors are equal to w with respect to $\equiv_{L_1}$. Moreover, $u \in v' \rightarrowtail_F (\{w\} \cup L_2)$. Thus, the inclusion holds. □

Compare this with Lemma 11. A weaker statement of this lemma "$(L_1 \rightarrowtail_F (L_2 \cup \{w\})) \cap L_3^c \neq \emptyset$ if and only if $(L_1 \rightarrowtail_F (L_2 \cup [w]_{\equiv_{L_1}})) \cap L_3^c \neq \emptyset$" is enough to reduce the existence of a solution to $L_1 \rightarrowtail_F X = L_3$ to that of its syntactic solution with respect to L_1. The algorithm to construct the set of all syntactic solutions to $L_1 \rightarrowtail_F X = L_3$ should be quite straightforward from the explanation in Section 5.1. Note that a prerequisite for $L_1 \rightarrowtail_F X = L_3$ to be solvable by this algorithm is that

the index of $\equiv_{L_1}$ be finite, that is, the regularity of L_1 (not that of L_3). This causes a difference between $L_1 \rightarrowtail_F X = L_3$ and its insertion variant in the conditions on L_1, F, L_3 under which our syntactic-congruence-oriented algorithms work. For instance, the algorithm for $L_1 \rightarrowtail_F X = L_3$ allows L_3 to be DPCM as long as L_1 and F are regular due to Theorem 9 and Proposition 4.

One should not overlook the fact that Lemma 14 is more than the analog of Lemma 11. A correct interpretation of this lemma is that the representatives of equivalence classes in $\Sigma^*/\equiv_{L_1}$ can achieve whatever the elements of these classes do in terms of p-schema-based deletion, not depending on the choice of representatives. Let us describe this formally as follows.

Lemma 15. *Let $L_1 \subseteq \Sigma^*$ be a language and F be a p-schema. Then for a language L_2 and a complete system of representatives $\mathfrak{R}(L_1)$ of $\Sigma^*/\equiv_{L_1}$, $L_1 \rightarrowtail_F L_2 = L_1 \rightarrowtail_F (\sigma_{L_1}(L_2) \cap \mathfrak{R}(L_1))$.*

Since the choice of $\mathfrak{R}(L_1)$ is arbitrary, Lemma 15 implies that all of the solutions to $L_1 \rightarrowtail_F X = L_3$ can be completely characterized by only the solutions which are a subset of some fixed complete system $\mathfrak{R}(L_1)$ of representatives of $\Sigma^*/\equiv_{L_1}$. Let us fix $\mathfrak{R}(L_1)$ to be the set of smallest words of each equivalence class according to the lexicographical order. Note that when L_1 is regular, we can effectively construct $\mathfrak{R}(L_1)$ from the minimal DFA for L_1.

Theorem 16. *For a regular language R_1, a regular p-schema F, $L_3 \in$ DPCM, and a complete system $\mathfrak{R}(R_1)$ of representatives of $\Sigma^*/\equiv_{R_1}$, the set of all solutions to $R_1 \rightarrowtail_F X = L_3$ which are a subset of $\mathfrak{R}(R_1)$ is effectively constructible.*

Lemma 15 and Theorem 16 have at least two corollaries of significance. The first is of syntactic solutions: from Lemma 15, it is evident that syntactic solutions to $L_1 \rightarrowtail_F X = L_3$ are no more than an image of its solution in $\mathfrak{R}(L_1)$ by the saturator σ_{L_1}. The second is of minimal solutions: due to Lemma 15, none of two words in a minimal solution to $L_1 \rightarrowtail_F X = L_3$ belong to the same equivalence class.

Corollary 17. *For a regular language R_1, a regular p-schema F, and $L_3 \in$ DPCM, the set of all syntactic solutions to $R_1 \rightarrowtail_F X = L_3$ is effectively constructible.*

Corollary 18. *For a regular language R_1, a regular p-schema F, and $L_3 \in$ DPCM, the set of all minimal solutions to $R_1 \rightarrowtail_F X = L_3$ modulo $\equiv_{R_1}$ is effectively constructible.*

As might be obvious already, another corollary of them is that the number of languages which can be obtained from a regular language R_1 by the deletion based on a given p-schema F is at most $|2^{\Sigma^*/\equiv_{R_1}}|$. If F is regular, then all of these resulting languages are not only regular but effectively constructible (Corollary 5). A special case of this result with the operation being sequential deletion is known [10].

We cannot say that we have got the most out of the decidability of equivalence in Theorem 9 for the proposed algorithm. Indeed, we can compare not only regular

languages but also DCM with DPCM. Due to this, the equation $L_1 \rightarrowtail_F X = L_3$ whose lefthand side is guaranteed to be no more than DPCM is of interest. Since the regularity of L_1 being essential at least for our algorithm, we should study weaker conditions on F than the regularity. This, however, involves difficulties because deleting even a word (i.e., a *singleton* language) from a regular language based on a DCM(1) p-schema can generate a non-DPCM language (Proposition 4 of [13]).

5.3. *Solving multiple-variables equations*

There is one thing which deserves explicit emphasis: the set of all candidates of syntactic solutions is solely determined by L_3 and does not depend on L_1 or F at all. This property paves the way to algorithms to solve two-variables language equations of the form $X \leftarrowtail_F Y = L_3$ and $L_1 \leftarrowtail_X Y = L_3$ under the assumption that L_3 be regular. The special instance of this problem with $\leftarrowtail$ being catenation has been investigated in the name of decomposition of regular languages and proved to be decidable [15, 17].

Note that if an equation $X \leftarrowtail_F Y = L_3$ has a solution (L_1, L_2), then $(L_1, \sigma_{L_3}(L_2))$ is also its solution. This means that if the equation has a solution (pair of languages), then it also has a solution whose second element is a union of equivalence classes in $\Sigma^*/\equiv_{L_3}$. In other words, in order to decide whether $X \leftarrowtail_F Y = L_3$ has a solution, it suffices to decide whether at least one of the one-variable equations obtained by substituting such unions of equivalence classes into $X \leftarrowtail_F Y = L_3$ as Y has a solution or not. As a whole, this paragraph works as an algorithm we have been looking for.

For regular languages R_1, R_3 and a regular p-schema F, this algorithm works effectively to solve $X \leftarrowtail_F Y = R_3$ ($R_1 \leftarrowtail_X Y = R_3$). We replace the second step of the previous algorithm with the process to solve $X \leftarrowtail_F ß[i] = R_3$ (resp. $R_1 \leftarrowtail_X ß[i] = R_3$) for each $1 \leq i \leq n$ using the Kari's technique based on the inverse operation mentioned previously.

Theorem 19. *For a regular language R_3 and a regular p-schema F, it is decidable whether the equation $X \leftarrowtail_F Y = R_3$ has a solution or not.*

Theorem 20. *For regular languages R_1, R_3, it is decidable whether the equation $R_1 \leftarrowtail_X Y = R_3$ has a solution or not.*

Using the algorithm to solve $L_1 \rightarrowtail_F X = L_3$ and the above approach, we can solve the problem of deciding whether the equation $L_1 \rightarrowtail_X Y = L_3$ has a solution or not provided both L_1 and L_3 are regular. Note that the regularity of L_3 is required (cf. Corollary 17) for the second step (inverse-operation-based approach to solve $L_1 \rightarrowtail_X ß[i] = L_3$) to work.

Theorem 21. *For regular languages R_1, R_3, it is decidable whether the equation $R_1 \rightarrowtail_X Y = R_3$ has a solution.*

Unlike p-schema-based insertions, our proposal does not work to solve the equation $X \rightarrowtail_F Y = L_3$. This is because in this case it is not L_3 but L_1 with respect to which the syntactic solutions of $L_1 \rightarrowtail_F Y = L_3$ to be considered.

5.4. *Inequalities with p-schema-based insertion*

The usage of the proposed algorithm is not exclusive to solving *equations*, but of use in solving *inequalities* such as $L_1 \leftarrowtail_F X \subseteq R_3$. In the investigation on such inequalities, our interest lies in their maximal solutions. Due to Lemma 11, maximal solutions to $L_1 \leftarrowtail_F X \subseteq R_3$ have to be unions of equivalence classes in $\Sigma^*/\equiv_{R_3}$. Hence, by replacing equality test in the step 2 with the inclusion test: "for each $1 \leq i \leq n$, test whether $L_1 \leftarrowtail_F ß[i]$ is a subset of R_3," we can modify the algorithm so as to solve the problem of finding maximal solutions to $L_1 \leftarrowtail_F X \subseteq R_3$. Note that inclusion test between regular languages is also decidable. As a corollary, for example, for a regular language R, we can compute the set of all languages X satisfying $X^* \subseteq R$ and maximal with this inclusion property since this inequality can be rewritten as $\{\lambda\} \leftarrowtail_{F_*} X \subseteq R_3$.

It should now be obvious that the further extension introduced in Section 5.3 is applicable to modify the above algorithm so as to solve language inequalities of the form $X \leftarrowtail_F Y \subseteq R_3$ and $R_1 \leftarrowtail_X Y \subseteq R_3$. We can easily prove that $X \leftarrowtail_F ß[i] \subseteq R_3$ has a unique maximal solution and it is represented as $(R_3^c \rightarrowtail_F ß[i])^c$.

5.5. *Undecidability of $L_1 \leftarrowtail_F X = L_3$ and $L_1 \rightarrowtail_F X = L_3$*

Having proposed the algorithm to solve the equation $L_1 \leftarrowtail_F X = L_3$ in the case of L_1, F, L_3 being regular, i.e., NCM(0), our interest shifts onto the extent to which we can weaken the hypothesis on the three languages involved. Actually, we shall prove that once one of them becomes NCM(1), then this problem immediately turns into undecidable. Note that the universe problem for NCM(1) is undecidable while the emptiness for this class is decidable [9].

Weakening the hypothesis on L_1 was considered in [12] for parallel insertion, i.e., $\leftarrowtail_{F_{\text{pins}}}$, and proved to be undecidable when L_1 is context-free even if X is limited to be a singleton language. For this purpose, using a special symbol $\natural \notin \Sigma$ (this special symbol will be employed in the following without being defined each time), they proved that for a context-free language L, $L\natural \leftarrowtail_{F_{\text{pins}}} X = \Sigma^*\natural$ has a solution if and only if $L = \Sigma^*$. It is clear that this proof works even if L is restricted to be in NCM(1). As a result, the next proposition holds.

Proposition 22. *For $L_1 \in$ NCM(1), a regular language R_3, and a syntactic regular p-schema F, it is undecidable whether $L_1 \leftarrowtail_F X = R_3$ has a solution or not.*

Next we consider the case when L_3, instead of L_1, is in NCM(1).

Proposition 23. *For a regular language R_1, $L_3 \in$ NCM(1), and a syntactic regular p-schema F, it is undecidable whether $R_1 \leftarrowtail_F X = L_3$ has a solution or not.*

Proof. Let $F = \lambda \times \Sigma^* \times \lambda$, which is syntactic and regular. For an arbitrary NCM(1) L, we claim that $\Sigma^* \leftarrow\!\!\!-_F X = \natural L \natural$ has a solution if and only if $L = \Sigma^*$. The converse implication is evident; if $L = \Sigma^*$, then $X = \{\natural\}$ is a solution. Indeed, we can see that this is the only one possible solution to this equation. If X contains a word which begins (ends) with $a \in \Sigma$, then $\Sigma^* \leftarrow\!\!\!-_F X$ can produce a word which begins (resp. ends) with a, and hence, X is not a solution to the equation. In the same way, we can see that any word in X cannot contain two $\natural$'s. Thus, if the equation has a solution, then its left-hand side becomes $\natural\Sigma^*\natural$ so that L has to be Σ^* for this equation to hold. □

What remains to be investigated is whether the NCM(1) p-schema brings the undecidability to solving $L_1 \leftarrow\!\!\!-_F X = L_3$. The proof of this can be found in [13].

Proposition 24. *For regular languages R_1, R_3 and a syntactic* NCM(1) *p-schema F, it is undecidable whether $R_1 \leftarrow\!\!\!-_F X = R_3$ has a solution or not.*

Let us keep investigating the undecidability of the existence of a right operand by switching the operation involved to p-schema-based deletion. Kari proved the undecidability of the problem of whether $L_1 X^{-1} = R_3$ has a solution, and its singleton variant, for a context-free language L_1 by reducing the universe problem to this problem, where $^{-1}$ is right quotient [10, 11]. Right quotient is the deletion based on the syntactic regular p-schema F_{cat}. Thus, we have the following result.

Proposition 25. *For $L_1 \in$ NCM(1), a regular language R_3, and a syntactic regular p-schema F, it is undecidable whether $L_1 \rightarrowtail_F X = R_3$ has a solution or not.*

The next proposition for the case of L_3 being in NCM(1) can be proved basically using the same idea as the one used in the proof of Proposition 23. Due to the space limitation, its proof is omitted.

Proposition 26. *For a regular language R_1, $L_3 \in$ NCM(1), and a syntactic regular p-schema F, it is undecidable whether $R_1 \rightarrowtail_F X = L_3$ has a solution or not.*

Proposition 27. *For regular languages R_1, R_3 and a syntactic* NCM(1) *p-schema F, it is undecidable whether the equation $R_1 \rightarrowtail_F X = R_3$ has a solution or not.*

Proof. Let h' be a homomorphism $\{0,1\}^* \to (\Sigma \cup \{\#\})^*$ defined as $h'(0) = \Sigma\#\Sigma\#$ and $h'(1) = \Sigma\Sigma\#\#$. Note that for an arbitrary λ-free NCM(1), it is undecidable whether the language is equal to Σ^+. Let F, G be p-schemata with $\psi(F) = \#\Sigma h'(L)$ and $\psi(G) = \#\Sigma$. $F \cup G$ is a syntactic NCM(1) p-schema.

Let \$ be a special symbol not in Σ. We claim that $[b\$\{abcb, cabb\}^* \cup (\Sigma^* \setminus \{b\})a] \rightarrowtail_{F \cup G} X = \$\{ac, ca\}^*$ has a solution if and only if $L = \{0,1\}^+$. The only one element in the first operand from which deleting X based on F can result in \$ is $b\$$, and hence, $b \in X$. If another word w is in X, then $a \in (wa \rightarrowtail_G X)$; however, a is not included in the right-hand side of this equation. Thus, the only one possible solution to this equation is $X = \{b\}$. Note that both $(\Sigma^* \setminus \{b\})a \rightarrowtail_F \{b\}$ and

$(\Sigma^* \setminus \{b\})a \rightarrowtail_G \{b\}$ are empty. Indeed, the emptiness of the first is due to the fact that any element of $\#\Sigma h'(L)$ ends with $\#$, but $a \neq b$. Hence, the equation holds if and only if $b\$\{abcb, cabb\}^* \leftarrowtail_{F \cup G} \{b\} = \$\{ac, ca\}^*$. The latter equation holds if and only if $L = \{0,1\}^+$. □

Acknowledgments

The authors appreciate anonymous referees for their carefully reading the earlier version of this paper and making valuable comments on them. This research was supported by Natural Sciences and Engineering Research Council of Canada Discovery Grant R2824A01, and Canada Research Chair Award to L. K.

References

[1] M. Anselmo and A. Restivo. On languages factorizing the free monoid. *International Journal of Algebra and Computations*, 6:413–427, 1996.

[2] B. S. Baker and R. V. Book. Reversal-bounded multipushdown machines. *Journal of Computer and System Sciences*, 8:315–332, 1974.

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

[4] M. Daley, O. Ibarra, and L. Kari. Closure and decidability properties of some language classes with respect to ciliate bio-operations. *Theoretical Computer Science*, 306:19–38, 2003.

[5] C. W. Dieffenbach and G. S. Dveksler, editors. *PCR Primer: A Laboratory Manual*. Cold Spring Harbor Laboratory Press, 2003.

[6] M. Domaratzki, G. Rozenberg, and K. Salomaa. Interpreted trajectories. *Fundamenta Informaticae*, 73:81–97, 2006.

[7] M. Domaratzki and K. Salomaa. Decidability of trajectory-based equations. *Theoretical Computer Science*, 345:304–330, 2005.

[8] S. Ginsburg. *Algebraic and Automata-Theoretic Properties of Formal Languages*. Elsevier Science Inc., New York, 1975.

[9] O. H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *Journal of the ACM*, 25:116–133, 1978.

[10] L. Kari. *On Insertion and Deletion in Formal Languages*. PhD thesis, University of Turku, Department of Mathematics, SF-20500 Turku, Finland, 1991.

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

[12] L. Kari, G. Rozenberg, and A. Salomaa. L systems. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 1, pages 253–328. Springer, 1997.

[13] L. Kari and S. Seki. Schema for parallel insertion and deletion. In Y. Gao, H. Lu, S. Seki, and S. Yu, editors, *DLT 2010*, volume 6224, pages 267–278, 2010.

[14] L. Kari and G. Thierrin. Contextual insertions/deletions and computability. *Information and Computation*, 131:47–61, 1996.

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

[16] M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3:114–125, 1959.

[17] A. Salomaa and S. Yu. On the decomposition of finite languages. In G. Rozenberg and W. Thomas, editors, *Developments in Language Theory*, pages 22–31, 1999.