Schema for parallel insertion and deletion *

Lila Kari and Shinnosuke Seki

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

Abstract. We propose a general framework for parallel insertion/deletion operations based on p-schemata. 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 a p-schema specifies how to split the given word into factors between which the insertion of the language will take place. Parallel deletion based on a p-schema is defined as an "inverse" operation of parallel insertion based on the p-schema. Several well-known language operations are particular cases of p-schema-based insertions or deletions: catenation, Kleene star, reverse catenation, sequential insertion, parallel insertion, insertion next to a given letter, contextual insertion, right and left quotient, sequential deletion, parallel deletion. Additional operations that can be defined using p-schemata include contextual parallel insertion, as well as parallel insertion (deletion) of exactly n words, at most n words, an arbitrary number of words. We also consider the decidability and undecidability of existence of solutions of language equations involving p-schema-based parallel insertion/deletion.

1 Introduction

Since Adleman's success [1] in solving the Directed Hamiltonian Path Problem purely by biological means, which threw new light on fundamental research on operations in formal language theory, various bio-operations have been intensively investigated. Examples include hairpin inversion [2], circular insertion/deletion [3], excisions of loop, hairpin, and double-loop [4], and contextual insertion/deletion [5], to name a few.

The fact that one can experimentally implement in the laboratory some variants of insertions and deletions into/from DNA sequences [6], and use these as the sole primitives for DNA computation, gives practical significance to the research on insertion and deletion. Contextual insertion and deletion are also of theoretical interest because they have been proved to be Turing-universal [5]. In this paper, we will parallelize contextual insertion and deletion. For words x and

* We thank anonymous referees for their valuable comments on the earlier version of this paper. In particular, we are indebted to one of them for the comparison between our framework and I-shuffle. This research was supported by The Natural Sciences and Engineering Research Council of Canada Discovery Grant R2824A01 and Canada Research Chair Award to L.K.

y, the (x, y)-contextual insertion of a language L into a word w [5] results in the language

$$\bigcup_{w_1, w_2 \text{ with } w=w_1xyw_2} w_1xLyw_2.$$

In other words, one considers all the possibilities of cutting w into two segments, such that the first segment ends with x and the second segment begins with y, and for each such possibility L is inserted between these segments. This operation suggests that for any positive integer n, an n-tuple $(w_1, w_2, \ldots, w_n)$ of words may be used to control the parallel insertion of $n-1$ instances of L into $w = w_1w_2 \cdots w_n$ to generate the language $w_1Lw_2L \cdots Lw_{n-1}Lw_n$. A set of such tuples is called a *parallel operation schema* or *p-schema* for short, and we call the parallel insertion thus determined *parallel insertion based on the p-schema.* A p-schema can be used to control not only parallel insertion but parallel deletion as well. 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 concatenates them to generate the word $u = u_1u_2 \cdots u_n$. As we shall see in Section 3, various well-known sequential as well as parallel operations (catenation, Kleene star, reverse catenation, sequential insertion, parallel insertion, insertion next to a given letter, contextual insertion, right and left quotient, sequential deletion, parallel deletion) are special instances of parallel operations based on p-schemata. Additional operations that can be defined using p-schemata are contextual parallel insertion, as well as parallel insertion (deletion) of *exactly n words, at most n words, an arbitrary number of words.*

Besides being proper generalizations of existing language operations, parallel operations based on p-schemata lead to some interesting results when studied in the context of language equations. Equations of the form $X_1 \diamond X_2 = X_3$ have been intensely studied in the literature, where $\diamond$ is a binary operation on languages, and some of X_1, X_2, X_3 are fixed languages, while the others are unknowns (see, e.g., [5, 7–14]). In this paper, we focus on such language equations with $\diamond$ being p-schema-based insertion or deletion. Since these two operations are parameterized by p-schemata, we can also consider the problem of deciding whether $L_1 \diamond_X L_2 = L_3$ has a solution, i.e., whether there exists a p-schema F such that parallelly inserting L_2 into (deleting from) L_1 based on F results in L_3.

In general, procedures do not exist for solving such equations when they involve a context-free language. Therefore, we focus on solving equations of the form (1) $X \leftarrow_F R_2 = R_3$, (2) $R_1 \leftarrow_X R_2 = R_3$, (3) $R_1 \leftarrow_F X = R_3$, and their p-schema-based deletion variants, where all of R_1, R_2, R_3, F are regular[1]. Among these equations, the equations of the first or second form can be solved using the technique of [14]. The application of this technique presumes the property that the union of all the solutions to the given equation is the unique maximal solution. As we shall see, the third-type equations do not have this property,

[1] by catenating words in a tuple of words via a special symbol #, we can naturally associate a set of tuples of words with a language, and as such we can establish a Chomsky-hierarchy for the sets of tuples of words.

that is, they may have multiple maximal solutions. Algorithms to solve these equations are one of the main contributions of this paper. Our algorithms work not only as a procedure to decide the existence of solutions, but as a procedure to enumerate all maximal solutions (Theorems 6 and 7). Moreover, combining these algorithms with the algorithms to solve the equations of the first or second form (outlined in Section 5) enables us to solve two-variables equations of the form $X \leftharpoonup_F Y = R_3$ (Theorem 9), $R_1 \leftharpoonup_X Y = R_3$ (Theorem 10), and $R_1 \rightharpoonup_X Y = R_3$ (Theorem 11). The proposed algorithms can be modified to also solve inequality (set inclusion) variants of the above-mentioned equations with maximality condition on variables.

2 Preliminaries

By Σ we denote a finite alphabet, and the set of words over Σ is denoted by Σ^* which includes the *empty word* λ. For a given word w, its length is denoted by $|w|$, and its reversal is denoted by w^R. For an integer $n \geq 0$, Σ^n, $\Sigma^{\leq n}$, and $\Sigma^{\geq n}$ denote the sets of all words of length *exactly* n, *at most* n, and *at least* n, respectively. 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. Let us denote the set of all prefixes (suffixes) of w by $\mathrm{Pref}(w)$ (resp. $\mathrm{Suf}(w)$). For a language $L \subseteq \Sigma^*$, $L^c = \Sigma^* \setminus L$.

Regular languages are specified by (non-deterministic) finite automata (NFA) $A = (Q, \Sigma, \delta, s, F)$, where Q is a finite set of states, $s \in Q$ is the start state, $F \subseteq Q$ is a set of final states, and δ is a map from $Q \times \Sigma$ to 2^Q. For notational convenience, we employ the notation NFA also to denote a language accepted by an NFA (we use this slight abuse of notation for other kinds of acceptors). The family of languages accepted by NFAs is denoted by REG. An NFA is said to be *deterministic* if δ is a function. The deterministic property of a machine is stated explicitly by using the capital letter D. A language is said to be *effectively regular* if there exists an algorithm to construct an NFA which accepts this language.

A characterization of languages can be given in terms of *syntactic semigroups*. 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). This is called the *syntactic congruence* of L, which is formally 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 $w \in \Sigma^*$, a set $[w]_{\equiv_L} = \{u \in \Sigma^* \mid w \equiv_L u\}$ is called an *equivalence class* with w as its representative. The number of equivalence classes is called the *index* of $\equiv_L$.

Theorem 1 ([15]). *Let $L \subseteq \Sigma^*$ be a language. The index of $\equiv_L$ is finite if and only if L is regular.*

For technical reasons, we define a function called *saturator with respect to a language* L_1. Let σ_{L_1} be a function from a word w into the equivalence class $[w]_{\equiv_{L_1}}$. The saturator w.r.t. L_1 is its extension defined as $\sigma_{L_1}(L) = \bigcup_{w \in L} [w]_{\equiv_{L_1}}$.

We can choose an arbitrary word in $[w]_{\equiv_L}$ as a representative of this class. By taking a representative from every class, we can construct a subset of Σ^* called a *complete system of representatives* of $\Sigma^*/\equiv_L$. In particular, for a regular language R, there exists a complete system of representatives which is computable. Let $A = (Q, \Sigma, \delta, s, F)$ be the (unique) minimal-DFA for R. Then $u \equiv_R v$ if and only if $\delta(q, u) = \delta(q, v)$ for any $q \in Q$. Hence, the index of $\equiv_L$ is at most $|Q|^{|Q|}$.

Theorem 2. *Let R be a regular language and $A = (Q, \Sigma, \delta, s, F)$ be the min-DFA for R. Each equivalence class in $\Sigma^*/\equiv_R$ is regular, and contains a word of length at most $|Q|^{|Q|}$.*

Corollary 1. *For a regular language R, there exists a computable complete system of representatives of $\Sigma^*/\equiv_L$.*

3 Parallel insertion and deletion schema

Imagine that we will insert a language L into a word u in parallel. Let $\prod_{i=1}^{n} \Sigma^*$ be the Cartesian product of Σ^* with itself n times; that is to say, the set of all n-tuples of words. Let $\mathfrak{F} = \bigcup_{n\geq 1} \underbrace{\Sigma^* \times \Sigma^* \times \cdots \times \Sigma^*}_{n \text{ times}}$. 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 $u_1 L u_2 L \cdots u_{n-1} L u_n$. The set can be also used to control a parallel deletion. For this intended end-usage, we call a subset of $\mathfrak{F}$ a *parallel schema*, or shortly *p-schema*, *over* Σ.

As abstracted above, a p-schema F enables us to define the *(parallel) insertion* $\leftarrowtail_F$ as: for a word $u \in \Sigma^*$ and a language $L \subseteq \Sigma^*$,

$$u \leftarrowtail_F L = \bigcup_{n\geq 1, u=u_1\cdots u_n, (u_1,\ldots,u_n)\in F} u_1 L u_2 L \cdots u_{n-1} L u_n.$$

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

$$w \rightarrowtail_G L = \{u_1 \cdots u_n \mid n \geq 1, x_1, \ldots, x_{n-1} \in L, \\ (u_1, \ldots, u_n) \in G, w = u_1 x_1 u_2 x_2 \cdots u_{n-1} x_{n-1} u_n\},$$

These operations are extended to languages in a conventional manner: for a language L_1, $L_1 \leftarrowtail_F L = \bigcup_{u\in L} u \leftarrowtail_F L$ and $L_1 \rightarrowtail_G L = \bigcup_{w\in L} w \rightarrowtail_G L$.

Many of the well-known operations are particular cases of p-schema-based operations. We list instances of p-schema-based insertion:

catenation	$F_{\text{cat}} = \Sigma^* \times \lambda$,
reverse catenation	$F_{\text{rcat}} = \lambda \times \Sigma^*$,
(sequential) insertion	$F_{\text{sins}} = \Sigma^* \times \Sigma^*$,
parallel insertion	$F_{\text{pins}} = \bigcup_{n\geq 0} (\lambda \times \prod_{i=1}^{n} \Sigma \times \lambda)$.

Deletions based on F_{cat}, F_{rcat}, F_{sins}, and F_{pins} correspond to right and left quotient, (sequential) deletion, and parallel deletion, respectively.

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

$$F_{\mathrm{pins}(n)} = \prod_{i=1}^{n+1} \Sigma^*, \qquad F_{\mathrm{pins}(\leq n)} = \bigcup_{i=0}^{n} F_{\mathrm{pins}(i)}, \qquad F_* = \bigcup_{i=0}^{\infty} F_{\mathrm{pins}(i)},$$

respectively. Using for instance F_*, one can implement Kleene-star, the most well-studied unary operation in formal language theory, as $L^* = \lambda \leftarrowtail_{F_*} L$.

The p-schemata introduced so far are "syntactic" in a sense, while many of semantic (letter-sensitive) operations are known. For a letter $b \in \Sigma$, *parallel insertion next to b* [13] is the insertion based on $F_{\mathrm{pinsb}} = \{(u_1, u_2, \ldots, u_n) \mid n \geq 1, u_1, \ldots, u_n \in (\Sigma \backslash \{b\})^* b\}$. For a context $C \subseteq \Sigma^* \times \Sigma^*$, *$C$-contextual (sequential) insertion* [5] is the insertion based on $F_{\mathrm{scins}}(C) = \bigcup_{(x,y)\in C} \Sigma^* x \times y\Sigma^*$. This operation is naturally parallelized as *C-contextual parallel insertion* with the p-schema $F_{\mathrm{pcins}}(C) = \{(u_1, \ldots, u_n) \mid n \geq 1, \forall 1 \leq i < n, \big(\mathrm{Suf}(u_i) \times \mathrm{Pref}(u_{i+1})\big) \cap C \neq \emptyset\}$.

It may be worth noting that the descriptional powers of our framework and of I-shuffle proposed by Domaratzki, Rozenberg, and Salomaa [16] (a generalization of semantic shuffle proposed by Domaratzki [11]) are incomparable. Indeed, only I-shuffle can specify contexts not only on the left operand but also on the right operand, while p-schema-based operations can insert/delete multiple copies of right operand. Thus, insertion/deletion based on a p-schema which contains 2-tuples and/or 1-tuples is a special instance of I-shuffle.

4 Hierarchy of p-schemata and closure properties

In this section, we investigate closure properties of abstract families of acceptors augmented with reversal-bounded counters under the p-schema-based operations. Such an acceptor was proposed by Ibarra [17] as the *counter machine*. For $k \geq 0$, let NCM(k) be the class of NFAs augmented with k reversal-bounded counters, and NCM be the union of such classes over all k's. By augmenting an NCM with an unrestricted pushdown stack, we obtain a *non-deterministic pushdown counter machine* (NPCM). For $k \geq 0$, let NPCM(k) be an NPCM with k reversal-bounded counters. DCM(k), DPCM(k), DCM, and DPCM are the deterministic analogs of NCM(k), NPCM(k), NCM, and NPCM. A desirable property specific to these deterministic classes is proved by Ibarra [17] as follows:

Theorem 3. *For $L_1 \in$ DCM and $L_2 \in$ DPCM, it is decidable whether $L_1 = L_2$.*

It is natural to encode a tuple $(u_1, u_2, \ldots, u_n)$ as a word $u_1 \# u_2 \# \cdots \# u_n$ using a special symbol $\#$. Denoting this (one-to-one) encoding by ψ, we can encode a p-schema F as $\psi(F) = \{\psi(f) \mid f \in F\}$. Furthermore, we say that a p-schema F is *in a language class $\mathcal{L}$* if $\psi(F) \in \mathcal{L}$. For instance, F is *regular* if $\psi(F)$ is a regular language over $\Sigma \cup \{\#\}$.

First of all, we prove that REG is closed under insertion/deletion based on a regular p-schema as Corollary 2. Actually, the following stronger result holds, though the rest of this paper does not require more than Corollary 2.

Proposition 1. *Let $L_1 \in \mathrm{NCM}(k_1)$, $L_2 \in \mathrm{REG}$, and F be a p-schema in $\mathrm{NCM}(k_\psi)$. Then both $L_1 \leftarrowtail_F L_2$ and $L_1 \rightarrowtail_F L_2$ are in $\mathrm{NCM}(k_1 + k_\psi)$.*

Proof. We show only a construction of an NCM M for $L_1 \leftarrowtail_F L_2$, and omit the construction of an NCM for $L_1 \rightarrowtail_F L_2$.

Let M_2 be a finite automaton for L_2, and M_1, M_ψ be respective NCMs with k_1, k_ψ counters for $L_1, \psi(F)$. The NCM M expects its input to be of the form $u_1x_1u_2x_2\cdots x_{n-1}x_n$ for some integer $n \geq 1$, $u_1u_2\cdots u_n \in L_1$, $x_1, x_2, \ldots, x_{n-1} \in L_2$, 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 M_2 on x_i after having M_ψ make a #-transition. When M_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 M_2 is initialized every time it is invoked. □

Corollary 2. *For regular languages R_1, R_2 and a regular p-schema F, both $R_1 \leftarrowtail_F R_2$ and $R_1 \rightarrowtail_F R_2$ are effectively regular.*

We can prove an analogous result of Proposition 1 for NPCM. By enlarging some of the respective language classes which L_1 and F belong to up to NPCM and a class which L_2 belongs to up to CFL, we can ask whether or not $L_1 \leftarrowtail_F L_2$ or $L_1 \rightarrowtail_F L_2$ are in NPCM. In the following we only address some non-closure properties of DPCM with implications to language equation solvability in the next section.

Let us define the *balanced language* L_b over $\Sigma = \{a, \$\}$ as follows:

$$L_b = \{a^{i_1}\$a^{i_2}\$\cdots\$a^{i_k}\$a^{i_{k+1}}\$\cdots\$a^{i_n} \mid n \geq 2,\ i_1, \ldots, i_n \geq 0 \text{ and } \exists 1 \leq k < n \text{ such that } i_1 + i_2 + \cdots + i_k = i_{k+1} + \cdots + i_n\}.$$

In other words, a word in L_b has a central marker \$ so that the number of a's to the left of this marker is equal to the number of a's to its right. For $L_1 = \{a^n\$a^n \mid n \geq 1\}$, we obtain $L_1 \leftarrowtail_{F_*} \$ = L_b$. Recall the definition of F_*; in this case it scatters an arbitrary number of \$'s into any word in L_1. We can generate L_b also by deletion. Let $L_1 = \bigcup_{n\geq 0}(a^n \sqcup\!\sqcup \$^*)\$\#(a^n \sqcup\!\sqcup \$^*)$ and $F = \{a, \$\}^* \times \{a, \$\}^*$, where $\sqcup\!\sqcup$ denotes shuffle operation. Then $L_b = L_1 \rightarrowtail_F \#$. These L_1's are DCM(1). L_b is clearly in NCM(1) because the non-determinism makes it possible for the reversal-bounded counter to guess when it should transit into its decrementing mode. In contrast, L_b is proved not to be DPCM (see, e.g., [18]). Consequently we have the following non-closure property.

Proposition 2. *There exist $L_1 \in \mathrm{DCM}(1)$, a regular p-schema F, and a singleton language L_2 such that $L_1 \leftarrowtail_F L_2 \notin \mathrm{DPCM}$.*

Proposition 3. *There exist $L_1 \in \mathrm{DCM}(1)$, a regular p-schema F, and a singleton language L_2 such that $L_1 \rightarrowtail_F L_2 \notin \mathrm{DPCM}$.*

By swapping the roles of L_1 and F in the above example, we can also obtain the following non-closure property.

Proposition 4. *There exist a regular language R_1, a singleton language L_2, and a $\mathrm{DCM}(1)$ p-schema F such that $R_1 \rightarrowtail_F L_2 \notin \mathrm{DPCM}$.*

5 Language equations with *p*-schemata-based operations

In this section, we consider language equations involving p-schema-based operations. The simplest equations to be studied are one-variable equations of the form $X \leftarrowtail_F L_2 = L_3$, $L_1 \leftarrowtail_X L_2 = L_3$, $L_1 \leftarrowtail_F X = L_3$, and their deletion variants. Such equations with special instances of p-schema-based operations (catenation, insertion, etc.) as well as incomparable operations (shuffle, etc.) have been intensively studied for the last decades [8, 9, 11–14, 19]. These papers mainly dealt with language equations with the property that the union of all their solutions (if any) is also their solution (maximum solution). For instance, if $XL = R$ and $YL = R$, then $(X \cup Y)L = R$. For such equations, we can employ a technique established in [14]; assuming a given equation has a solution, firstly construct the candidate of its maximum solution, and then substitute it into the equation to check whether it is actually a solution. Since $X \leftarrowtail_F L_2 = L_3$, $L_1 \leftarrowtail_X L_2 = L_3$, and their deletion variants have this property, this technique can solve these equations. We will now see how to construct the candidate for each.

In [9], Cui, Kari, and Seki defined the left-l-inverse relation between operations as: the operation $\bullet$ is *left-l-inverse* of the operation $\circ$ if for any words $u, w \in \Sigma^*$ and any language $L \subseteq \Sigma^*$, $w \in u \circ L \iff u \in w \bullet L$. This is a symmetric relation. By definition, insertion and deletion based on the same p-schema are left-l-inverse to each other. There they proved that for operations $\circ, \bullet$ which are left-l-inverse to each other, if $X \circ L_2 = L_3$ has a solution, then $(L_3^c \bullet L_2)^c$ is its maximum solution.

Theorem 4. *For regular languages R_2, R_3 and a regular p-schema F, the existence of a solution to both $X \leftarrowtail_F R_2 = R_3$ and $X \rightarrowtail_F R_2 = R_3$ is decidable.*

Proof. Both $(R_3^c \rightarrowtail_F R_2)^c \leftarrowtail_F R_2$ and $(R_3^c \leftarrowtail_F R_2)^c \rightarrowtail_F R_2$ are regular according to Corollary 2 and the fact that REG is closed under complement. Now it suffices to employ Theorem 3 for testing the equality. □

For $L_1 \leftarrowtail_X L_2 = L_3$, the candidate is $F_{\max} = \{f \in \mathfrak{F} \mid L_1 \leftarrowtail_f L_2 \subseteq L_3\}$. For $L_1 \rightarrowtail_X L_2 = L_3$, $F_{\max}$ should be rather $\{f \in \mathfrak{F} \mid L_1 \rightarrowtail_f L_2 \subseteq L_3\}$. When L_1, L_2, L_3 are all regular, we can construct an NFA for $\psi(\mathfrak{F} \setminus F_{\max})$, which is equal to $(\Sigma \cup \#)^* \setminus \psi(F_{\max})$. A similar problem was studied in [12], and our construction originates from theirs. As such, the proof of next result is omitted.

Theorem 5. *For regular languages R_1, R_2, R_3, the existence of a solution to both $R_1 \leftarrowtail_X R_2 = R_3$ and $R_1 \rightarrowtail_X R_2 = R_3$ is decidable.*

5.1 Solving $L_1 \leftharpoonup_F X = L_3$

In contrast, the equations $L_1 \leftharpoonup_F X = L_3$ and $L_1 \rightharpoonup_F X = L_3$ may not have a maximum solution. For example, let $L_1 = L_3 = \{a^{2n} \mid n \geq 1\}$, and $F = F_{\text{pins}(2)} \cup F_{\text{pins}(0)}$. Both $L_{\text{even}} = \{a^{2m} \mid m \geq 0\}$ and $L_{\text{odd}} = \{a^{2m+1} \mid m \geq 0\}$ are (maximal) solutions to $L_1 \leftharpoonup_F X = L_3$. On the other hand, $L_1 \leftharpoonup_F (L_{\text{even}} \cup L_{\text{odd}})$ can generate a^3, which is not in L_3. For deletion, let $F = \{(\lambda, aba), (\lambda, \lambda, \lambda), (aba, \lambda)\}$, and $L_1 = \{ababa\}$. Then $L_1 \rightharpoonup_F \{ab\} = L_1 \rightharpoonup_F \{ba\} = \{aba\}$, but $L_1 \rightharpoonup_F \{ab, ba\} = \{aba, a\}$. These examplify that we cannot apply the previously-mentioned approach to solving language equations with the second operand being unknown.

We propose an alternative approach based on an idea from Conway (Chapter 6 of [8]) 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. The idea shall be briefly explained in terms of p-schema-based operations in order to step into more general cases than the case when all the involved languages are regular.

Lemma 1. *Let L, L_1 be languages. Then $(L_1 \leftharpoonup_F (L_2 \cup w)) \cap L \neq \emptyset$ if and only if $(L_1 \leftharpoonup_F (L_2 \cup [w]_{\equiv_L})) \cap L \neq \emptyset$ for any word w and language L_2.*

By replacing L in this lemma with L_3^c, we can see that if $L_1 \leftharpoonup_F (L_2 \cup w) \subseteq L_3$, then $L_1 \leftharpoonup_F (L_2 \cup [w]_{\equiv_{L_3}}) \subseteq L_3$. Thus, it makes sense to introduce the notion of a syntactic solution. For 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 $\Sigma^* / \equiv_L$.

Proposition 5. *For languages L_1, L_3, the equation $L_1 \leftharpoonup X = L_3$ has a solution if and only if it has a syntactic solution with respect to L_3.*

Thus, in order to determine whether $L_1 \leftharpoonup_F X = L_3$ has a solution, it suffices to test whether it has a syntactic solution. On condition that this test can be executed, this problem becomes decidable. If L_3 is regular, then the number of candidates of syntactic solution is finite (Theorem 1), and they are regular (Theorem 2). Let $ß = \{\sigma_{R_3}(L) \mid L \subseteq \Sigma^*\}$, the set of all candidates of syntactic solution. A pseudocode to solve $L_1 \leftharpoonup_F X = R_3$ is given below:

Algorithm to solve $L_1 \leftharpoonup_F X = R_3$

1. Order the elements of ß 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 $L_1 \leftharpoonup_F ß[i]$ is equal to R_3.

With the further condition that L_1 and F are chosen so that any language obtained by substituting a candidate into $L_1 \leftharpoonup_F X$ is comparable with R_3 for equality, this algorithm becomes executable. One such condition of significance is that both L_1 and F are regular. In this case, the algorithm, Theorem 3, and Corollary 2 lead us to the next theorem, which is stronger than decidability. It should be noted that maximal solutions are syntactic.

Theorem 6. *For regular languages R_1, R_3 and a regular p-schema F, the set of all syntactic solutions to $R_1 \leftarrowtail_F X = R_3$ is computable.*

The regularity of R_3 is necessary for the algorithm to work, whereas such condition is not imposed on L_1. If a condition on L_1, F under which $L_1 \leftarrowtail_F$ ß$[i] \in$ DPCM for any $1 \leq i \leq |$ß$|$ were found, we could solve $L_1 \leftarrowtail_F X = R_3$ under it using Theorem 3. This is an unsettled question, but as suggested in Proposition 2, weakening the condition on L_1 slightly can make $L_1 \leftarrowtail_F X$ non-DPCM. It is probably more promising to broaden the class of F.

5.2 Solving $L_1 \rightarrowtail_F X = L_3$

Let us continue the investigation on the existence of right operand by changing the operation to p-schema-based deletion.

Lemma 2. *Let L_1 be a language. Then $L_1 \rightarrowtail_F (\{w\} \cup L_2) = L_1 \rightarrowtail_F ([w]_{\equiv_{L_1}} \cup L_2)$ for any word w, language L_2, and a p-schema F.*

Proof. Let $u \in L_1 \rightarrowtail_F ([w]_{\equiv_{L_1}} \cup L_2)$; that is, 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 the syntactic congruence of L_1. Moreover, $u \in v' \rightarrowtail_F (\{w\} \cup L_2)$. Thus, $L_1 \rightarrowtail_F (\{w\} \cup L_2) \supseteq L_1 \rightarrowtail_F ([w]_{\equiv_{L_1}} \cup L_2)$. □

This lemma provides us with two approaches to determine whether a given equation with p-schema-based deletion has a solution. The first approach is based on syntactic solutions. Given a language L_2, Lemma 2 implies that $L_1 \rightarrowtail_F L_2 = L_1 \rightarrowtail_F \sigma_{L_1}(L_2)$. Therefore, as in the case of insertion, the existence of a solution to $L_1 \rightarrowtail_F X = L_3$ is reduced to that of its syntactic solutions, but with respect to L_1 (not L_3). Moreover, maximal solutions are syntactic.

Proposition 6. *For languages L_1, L_3 and a p-schema F, the equation $L_1 \rightarrowtail_F X = L_3$ has a solution if and only if it has a syntactic solution with respect to L_1. Furthermore, its maximal solution (if any) is syntactic.*

With a straightforward modification, the algorithm presented in Sect. 5.1 can be used to output all syntactic solutions to $R_1 \rightarrowtail_F X = L_3$ with F being a regular p-schema. Thus, we have the following result, analogous to Theorem 6.

Theorem 7. *For a regular language R_1, $L_3 \in$ DPCM, and a regular p-schema F, the set of all syntactic solutions to $R_1 \rightarrowtail_F X = L_3$ is computable.*

Note that even if L_3 is DPCM, the equation above is solvable due to Corollary 2 and Theorem 3.

The existence of the second approach provided by Lemma 2 is due to the essential difference between Lemma 2 and its analog for insertion (Lemma 1).

A word obtained by deleting some words in L_2 from a word in L_1 can be also obtained by deleting their representatives in a complete system of representatives with respect to L_1 from the word in L_2 based on the same schema; this is not true for insertion. Since its choice is arbitrary, we fix $\mathfrak{R}(L_1)$ to be the set of smallest words according to the lexicographical order in each equivalence class. We say that a solution to $L_1 \rightarrowtail_F X = L_3$ is *representative* if it is a subset of $\mathfrak{R}(L_1)$.

Proposition 7. *For languages L_1, L_3 and a p-schema F, the equation $L_1 \rightarrowtail_F X = L_3$ has a solution if and only if it has a representative solution.*

If L_1 is regular, then $\mathfrak{R}(L_1)$ is a finite computable set due to Theorem 1 and Corollary 1, and hence, our argument based on representative solution amounts to the second approach.

Theorem 8. *For a regular language R_1, $L_3 \in$ DPCM, and a regular p-schema F, the set of all representative solutions of $R_1 \rightarrowtail_F X = L_3$ is computable.*

With Theorem 1, Lemma 2 also leads us to a corollary about the number of distinct languages obtained by p-schema-based deletion from a regular language. Namely, given a regular language R_1 and a p-schema F, there exist at most a finite number of languages which can be represented in the form $R_1 \rightarrowtail_F L_2$ for some language L_2. This result is known for sequential deletion [13].

5.3 Solving two-variables language equations and inequalities

There is one thing which deserves explicit emphasis: the set of all candidates of syntactic solutions is solely determined by only one of L_3, L_1, and does not depend on the other or F at all. This property paves the way to solving two-variables language equations of the form $X \leftarrowtail_F Y = L_3$, $L_1 \leftarrowtail_X Y = L_3$, and $L_1 \rightarrowtail_X Y = L_3$. The first equation with $F = F_{\mathrm{cat}}$ (catenation) has been investigated under the name of *decomposition of regular languages* and proved to be decidable [19, 20].

Let us assume that (L_1, L_2) is a solution of $X \leftarrowtail_F Y = L_3$. Then $\sigma_{L_3}(L_2)$ is a solution of $L_1 \leftarrowtail_F Y = L_3$, and hence, $(L_1, \sigma_{L_3}(L_2))$ is also a solution of $X \leftarrowtail_F Y = L_3$. This means that if the equation has a solution (pair of languages), then it also has a solution whose second element is a sum of equivalence classes in $\Sigma^*/\equiv_{L_3}$. Therefore, solving $X \leftarrowtail_F ß[i] = L_3$ for all $1 \leq i \leq |ß|$ using Theorem 4 amounts to solving the two-variables equation. For a regular language R_3 and a regular p-schema F, the above method works effectively to solve $X \leftarrowtail_F Y = R_3$.

Theorem 9. *It is decidable whether the equation $X \leftarrowtail_F Y = R_3$ has a solution or not if both R_3 and F are regular.*

Undertaking the same "two-staged" strategy but using Theorem 5 instead, we can solve the equations of second and third forms.

Theorem 10. *For regular languages R_1, R_3, it is decidable whether the equation $R_1 \leftarrow_X Y = R_3$ has a solution or not.*

Theorem 11. *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 insertion, this strategy does not work to solve the equation of the form $X \rightarrowtail_F Y = L_3$. This is because in this case it is not L_3 but L_1 that determines the syntactic solutions of $L_1 \rightarrowtail_F Y = L_3$.

The usage of the proposed algorithm is not exclusive to solving language *equations*. By replacing the equality test in Step 2 with the following inclusion test "for each $1 \leq i \leq |ß|$, test whether $L_1 \leftarrow_F ß[i]$ is a subset of R_3", the proposed algorithm can answer the problem of finding maximal solutions to the language inequality $L_1 \leftarrow_F X \subseteq R_3$, and with the two-staged strategy, this further enables us to solve $X \leftarrow_F Y \subseteq R_3$ and $L_1 \leftarrow_X Y \subseteq R_3$. Now it should be trivial how to approach $R_1 \rightarrowtail_F X \subseteq L_3$ and $R_1 \rightarrowtail_X Y \subseteq L_3$.

5.4 Undecidability

We conclude this section and this paper by complementing the decidability results obtained so far with some undecidability results for one-variable equations. Usually, the existence of solutions to a language equation of this type is decidable if all known languages are regular, and undecidable if at least one of the known languages is context-free. The results of this section bring down, for several cases, the limit for undecidability of existence of solutions of such language equations from the class of context-free languages to NCM(1). The equation $L_1 \leftarrow_F X = L_3$ is solvable in the case of L_1, F, L_3 being regular, i.e., NCM(0). Actually, we shall prove that once one of them becomes NCM(1), then this problem immediately turns into undecidable.

Proposition 8. *For languages L_1, L_3 and a p-schema F, if one of L_1, L_3, F is in* NCM(1) *and the others are regular, it is undecidable whether $L_1 \leftarrow_F X = L_3$ has a solution or not.*

Proof. We employ the reduction of universe problem (whether a given NCM(1) is Σ^*) into these problems. The universe problem is known to be undecidable for the class NCM(1) [17]. Because of space limitations, we can consider here only the case when F is an NCM(1) p-schema.

Let $\natural, \$$ be special symbols not included in Σ. Based on a given $L \in$ NCM(1), we define a p-schema $F = \lambda \times \$L$, which is in NCM(1), too. Then for regular languages $\$\Sigma^*$ and $\natural\$\Sigma^*$, we claim that $\$\Sigma^* \leftarrow_F X = \natural\$\Sigma^*$ has a solution $\Longleftrightarrow$ $L = \Sigma^*$. Indeed, the left-hand side of the above equation is $X\$L$ so that its only one possible solution is $X = \natural$. Thus, the existence of the solution leads us immediately to that L is universe. □

For the equation $L_1 \rightarrowtail_F X = L_3$, the similar undecidability result holds.

Proposition 9. *For languages L_1, L_3 and a p-schema F, if one of L_1, L_3, F is in* NCM(1) *and the others are regular, it is undecidable whether $L_1 \rightarrowtail_F X = L_3$ has a solution or not.*

References

1. Adleman, L.M.: Molecular computation of solutions to combinatorial problems. Science **266**(5187) (November 1994) 1021–1024
2. Ehrenfeucht, A., Harju, T., Petre, I., Rozenberg, G.: Patterns of micronuclear genes in ciliates. In Jonoska, N., Seeman, N.C., eds.: DNA 7. Volume 2340 of Lecture Notes in Computer Science., Springer (2002) 279–289
3. Landweber, L.F., Kari, L.: The evolution of cellular computing: Nature's solution to a computational problem. In Kari, L., Rubin, H., Wood, D., eds.: Proc. DNA-Based Computers IV. (1999) 3–13
4. Freund, R., Martín-Vide, C., Mitrana, V.: On some operations on strings suggested by gene assembly in ciliates. New Generation Computing **20** (2002) 279–293
5. Kari, L., Thierrin, G.: Contextual insertions/deletions and computability. Information and Computation **131** (1996) 47–61
6. Dieffenbach, C.W., Dveksler, G.S., eds.: PCR Primer: A Laboratory Manual. Cold Spring Harbor Laboratory Press (2003)
7. Anselmo, M., Restivo, A.: On languages factorizing the free monoid. International Journal of Algebra and Computations **6** (1996) 413–427
8. Conway, J.H.: Regular Algebra and Finite Machines. Chapman and Hall, London (1971)
9. Cui, B., Kari, L., Seki, S.: Block insertion and deletion on trajectories. In preparation (2009)
10. Daley, M., Ibarra, O., Kari, L.: Closure and decidability properties of some language classes with respect to ciliate bio-operations. Theoretical Computer Science **306** (2003) 19–38
11. Domaratzki, M.: Semantic shuffle on and deletion along trajectories. In Calude, C.S., Claude, E., Dinneen, M.J., eds.: DLT 2004. Volume 3340 of Lecture Notes in Computer Science., Springer (2004) 163–174
12. Domaratzki, M., Salomaa, K.: Decidability of trajectory-based equations. Theoretical Computer Science **345** (2005) 304–330
13. Kari, L.: On Insertion and Deletion in Formal Languages. PhD thesis, University of Turku, Department of Mathematics, SF-20500 Turku, Finland (1991)
14. Kari, L.: On language equations with invertible operations. Theoretical Computer Science **132** (1994) 129–150
15. Rabin, M.O., Scott, D.: Finite automata and their decision problems. IBM Journal of Research and Development **3** (1959) 114–125
16. Domaratzki, M., Rozenberg, G., Salomaa, K.: Interpreted trajectories. Fundamenta Informaticae **73** (2006) 81–97
17. Ibarra, O.H.: Reversal-bounded multicounter machines and their decision problems. Journal of the ACM **25** (1978) 116–133
18. Chiniforooshan, E., Daley, M., Ibarra, O.H., Kari, L., Seki, S.: Reversal-bounded counter machines and multihead automata: Revisited. in preparation (2010)
19. Kari, L., Thierrin, G.: Maximal and minimal solutions to language equations. Journal of Computer and System Sciences **53** (1996) 487–496
20. Salomaa, A., Yu, S.: On the decomposition of finite languages. In Rozenberg, G., Thomas, W., eds.: Developments in Language Theory. (1999) 22–31