Available online at www.sciencedirect.com

SCIENCE DIRECT®

Journal of Computer and System Sciences 70 (2005) 157–178

JOURNAL OF COMPUTER AND SYSTEM SCIENCES

www.elsevier.com/locate/jcss

Language equations, maximality and error-detection☆

Lila Kari[a], Stavros Konstantinidis[b,*]

[a]*Department of Computer Science, University of Western Ontario, London, Ont., Canada N6A 5B7*
[b]*Department of Mathematics and Computing Science, Saint Mary's University, Halifax, NS, Canada B3H 3C3*

Received 17 July 2003; received in revised form 25 August 2004

Abstract

We use some 'natural' language operations, such as shuffle (scattered insertion) and scattered deletion to model noisy channels, that is, nondeterministic processes transforming words to words. In this spirit, we also introduce the operation of scattered substitution and derive the closure properties of the language families in the Chomsky hierarchy under this operation. Moreover, we consider a certain type of language inequations involving language operations and observe that, by varying the parameters of such an inequation, we can define families of codes such as prefix and infix, as well as families of error-detecting languages. Our results on this type of inequations include a characterization of the maximal solutions, which provides a uniform method for deciding whether a given regular code of the type defined by the inequation is maximal.
© 2004 Elsevier Inc. All rights reserved.

Keywords: Language operations; Language inequations; Closure properties; Maximal codes; Error detection

1. Introduction

Language operations, such as catenation, shuffle (scattered insertion) and scattered deletion, have been a classical topic of study in formal language theory. In particular, the closure properties of language families in the Chomsky hierarchy under such operations are one of the central themes in this theory [13,7]. More recently, also the topic of language equations involving language operations other than catenation has

☆ Research partially supported by Grants R2824A01 and R220259 of the Natural Sciences and Engineering Research Council of Canada.

* Corresponding author.

E-mail addresses: lila@csd.uwo.ca (L. Kari), stavros@cs.stmarys.ca, s.konstantinidis@stmarys.ca (S. Konstantinidis).

doi:10.1016/j.jcss.2004.08.005

been of interest [8,9] (see [2] for language equations involving the catenation operation). In this work, we observe that certain language operations—in particular shuffle and scattered deletion—can be used to model noisy channels (in the sense of [11]). In this spirit we introduce another 'natural' language operation, the operation of scattered substitution, and derive the closure properties of the language families in the Chomsky hierarchy under this operation. We also observe that a certain type of language inequations can be used to define code-related properties of languages. More specifically, consider the inequation

$$X \diamond L \subseteq X^c \quad \text{with} \quad X \subseteq M, \tag{*}$$

where X is the unknown language, X^c is the complement of X, L and M are fixed languages, and $\diamond$ is a binary language operation. Depending on the choice of $\diamond$, L, and M, the solution set of such an inequation could be the family of all prefix codes, hypercodes, infix codes, etc. (see [5] for such families of codes). Moreover, the pair $(\diamond, L)$ can be used to define a noisy channel, which we denote by $[\diamond L_\lambda]$. With this interpretation, the solution set of the inequation is the set of all languages that are error-detecting for the channel $[\diamond L_\lambda]$. Following certain ideas in [8,9] about language equations, we obtain a characterization of the maximal solutions of the inequation $(*)$, when $(*)$ is of type (c)—see Section 6. This yields a method for deciding whether a given regular code of the type defined by the inequation is maximal. We note that uniform methods for deciding code-related properties of regular languages have been considered in [6,4]. However, to our knowledge, there is no analogous uniform method for deciding the maximality property.

The paper is structured as follows. In the next section we provide the basic notation and background about formal languages, binary relations, word operations, language equations and error-detection. In Section 3 we give examples to demonstrate that certain code-related properties are definable via language inequations of type $(*)$. For the case of error-detection properties we need the concept of noisy channel. We show how to model certain channels using language operations in Section 4. In Section 5, we study the closure properties of language families in the Chomsky hierarchy under the operations involved in modelling channels with substitution errors. In Section 6 we point out the connection between error-detecting languages and the solutions of the above inequation and establish basic results about the maximal solutions of this inequation. When the inequation is of type (c) we obtain a necessary and sufficient condition for whether a given solution is maximal—see Corollary 6.7. In the last section we discuss some special cases and applications of our results. In particular, we show that (i) for certain inequations with finitely many maximal solutions there is a method for obtaining those solutions; (ii) the problem of whether the inequation has a solution of at least k elements, for some given k, is NP-complete; (iii) there are simple and efficient algorithms for deciding whether a given regular prefix code, or finite bifix code, or finite infix code, or fixed-length 1-error-detecting code is maximal.

2. Definitions, notations and background

2.1. Alphabet, word, language, automaton, binary relation

An *alphabet* is a finite and nonempty set of symbols. In the sequel we shall use a fixed alphabet Σ. The set of all words (over Σ) is denoted by Σ^*. This set includes the *empty word* λ. The length of a word w is denoted by $|w|$. For a nonnegative integer n and a word w, we use w^n to denote the word that consists of n

concatenated copies of w. The *Hamming distance* $H(u, v)$ between two words u and v of the same length is the number of corresponding positions in which u and v differ. For example, $H(abba, aaaa) = 2$.

A language L is a set of words, or equivalently a subset of Σ^*. A language is said to be λ-free if it does not contain the empty word. For a language L, we write L_λ to denote $L \cup \{\lambda\}$. If n is a nonnegative integer, we write L^n for the language consisting of all words of the form $w_1 \cdots w_n$ such that each w_i is in L. We also write L^* for the language $L^0 \cup L^1 \cup L^2 \cup \cdots$ and L^+ for the language $L^* - \{\lambda\}$. The notation L^c represents the complement of the language L; that is, $L^c = \Sigma^* - L$. For the classes of regular, context-free, and context sensitive languages, we use the notations REG, CF and CS, respectively.

A nondeterministic finite automaton with λ productions (or transitions), a *λ-NFA* for short, is a quintuple $A = (S, \Sigma, s_0, F, P)$ such that S is the finite and nonempty set of states, s_0 is the start state, F is the set of final states, and P is the set of productions of the form $sx \to t$, where s and t are states in S, and x is either a symbol in Σ or the empty word. If there is no production with $x = \lambda$, the automaton is called an *NFA*. If for every two productions of the form $sx_1 \to t_1$ and $sx_2 \to t_2$ of an NFA we have that $x_1 \neq x_2$ then the automaton is called a *DFA* (deterministic finite automaton). The language accepted by the automaton A is denoted by $L(A)$. The automaton is called *trim* if every state is reachable from the start state and can reach a final state in F (when $F \neq \emptyset$). The *size* $|A|$ of the automaton A is the number $|S| + |P|$. Note that the number $|S|$ of states of a trim automaton is at most $1 + |P|$; therefore, the size of such an automaton is $|A| = \Theta(|P|)$.

A *finite transducer* (in standard form) is a sextuple $T = (S, \Sigma, \Sigma', s_0, F, P)$ such that Σ' is the output alphabet, the components S, s_0, F are as in the case of λ-NFAs, and the set P consists of productions of the form $sx \to yt$ where s and t are states in S, $x \in \Sigma \cup \{\lambda\}$ and $y \in \Sigma' \cup \{\lambda\}$. If x is nonempty for every production then the transducer is called a *gsm* (generalized sequential machine). If, in addition, y is nonempty for every production then the transducer is called a *λ-free gsm*. The *relation realized by* the transducer T is denoted by $R(T)$. The concept of a trim transducer is the same as that in the case of automata. The size $|T|$ of the transducer T (in standard form) is $|S| + |P|$. Again, when the transducer is trim its size is $|T| = \Theta(|P|)$.

A *binary relation* γ, say, over Σ is a subset of $\Sigma^* \times \Sigma^*$. The *domain* of γ, denoted dom (γ), is the set of all words u such that (u, v) is in γ for some word v. We shall use the notation $\gamma(u)$ for the set $\{v \mid (u, v) \in \gamma\}$. This notation is extended to languages L as follows: $\gamma(L) = \cup_{u \in L} \gamma(u)$. The symbol γ^{-1} represents the inverse of the relation γ, which is equal to $\{(v, u) \mid (u, v) \in \gamma\}$. The *composition* $\gamma_1 \circ \gamma_2$ of two binary relations γ_1 and γ_2 is the binary relation $\{(u, v) \mid (u, z) \in \gamma_2$ and $(z, v) \in \gamma_1$, for some word $z\}$. A relation is called *rational* if it can be realized by a finite transducer.

We refer the reader to [14] or [16] for details on automata and formal languages.

2.2. Binary word operations

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^*. The *domain* of $\diamondsuit$, denoted dom $(\diamondsuit)$, is the set of all pairs (u, v) of words such that the set $u\diamondsuit v$ is not empty. The *left domain* of $\diamondsuit$ is dom$_1(\diamondsuit) = \{u : (u, v) \in$ dom $(\diamondsuit)$ for some word $v\}$. Similarly, the *right domain* of $\diamondsuit$ is dom$_2(\diamondsuit) = \{v : (u, v) \in$ dom $(\diamondsuit)$ for some word $u\}$. The *image* of $\diamondsuit$ is im $(\diamondsuit) = \bigcup_{u,v \in \Sigma^*} u\diamondsuit v$. The *characteristic relation* of $\diamondsuit$ is

$$C_\diamondsuit = \{(w, u, v) : w \in u\diamondsuit v\}.$$

For any languages X and Y, $X\diamond Y = \bigcup_{u\in X, v\in Y} u\diamond v$. It should be noted that every subset B of $\Sigma^* \times \Sigma^* \times \Sigma^*$ defines a unique binary word operation whose characteristic relation is exactly B.

Definition 2.1 (Kari [8]). Let $\diamond$ be an operation. The left inverse $\diamond^l$ of $\diamond$ is defined as

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

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

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

Definition 2.2. Let $\diamond$ be a binary word operation. The word operation $\diamond'$ defined by $u\diamond' v = v\diamond u$ is called *reversed* $\diamond$.

It should be clear that, for every binary operation $\diamond$, the triple (w, u, v) is in $C_\diamond$ if and only if (u, w, v) is in $C_{\diamond^l}$ if and only if (v, u, w) is in $C_{\diamond^r}$ if and only if (w, v, u) is in $C_{\diamond'}$. If x and y are symbols in $\{l, r, '\}$, the notation $\diamond^{xy}$ represents the operation $(\diamond^x)^y$. Using the above observations, one can establish identities between operations of the form $\diamond^{xy}$. For example, $\diamond^{ll} = \diamond^{rr} = \diamond'' = \diamond$ and $\diamond'^l = \diamond^{r'} = \diamond^{lr}$.

Next we list a few binary word operations together with their left and right inverses [7,8].

Catenation:[1] $u \cdot v = \{uv\}$, with $\cdot^l = \longrightarrow_{rq}$ and $\cdot^r = \longrightarrow_{lq}$.

Left quotient: $u \longrightarrow_{lq} v = \{w\}$ if $u = vw$, with $\longrightarrow^l_{lq} = \cdot'$ and $\longrightarrow^r_{lq} = \longrightarrow_{rq}$.

Right quotient: $u \longrightarrow_{rq} v = \{w\}$ if $u = wv$, with $\longrightarrow^l_{rq} = \cdot$ and $\longrightarrow^r_{rq} = \longrightarrow_{lq}$.

Insertion: $u \longleftarrow v = \{u_1vu_2 \mid u = u_1u_2\}$, with $\longleftarrow^l = \longrightarrow$ and $\longleftarrow^r = \rightleftharpoons'$.

Deletion: $u \longrightarrow v = \{u_1u_2 \mid u = u_1vu_2\}$, with $\longrightarrow^l = \longleftarrow$ and $\longrightarrow^r = \rightleftharpoons$.

Dipolar deletion: $u \rightleftharpoons v = \{w \mid u = v_1wv_2, v = v_1v_2\}$, with $\rightleftharpoons^l = \longleftarrow'$ and $\rightleftharpoons^r = \longrightarrow$.

Shuffle (*or scattered insertion*): $u \amalg v = \{u_1v_1 \cdots u_kv_ku_{k+1} \mid k \geqslant 1, u = u_1 \cdots u_ku_{k+1}, v = v_1 \cdots v_k\}$, with $\amalg^l = \rightsquigarrow$ and $\amalg^r = \rightsquigarrow'$.

Scattered deletion: $u \rightsquigarrow v = \{u_1 \cdots u_ku_{k+1} \mid k \geqslant 1, u = u_1v_1 \cdots u_kv_ku_{k+1}, v = v_1 \cdots v_k\}$, with $\rightsquigarrow^l = \amalg$ and $\rightsquigarrow^r = \rightsquigarrow$.

2.3. Language equations

The process of solving language equations has much in common with the process of solving algebraic equations. For example the equation $X\diamond L = R$ 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 $L\diamond Y = R$, 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.

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

Theorem 2.3. *Let $L, R \subseteq \Sigma^*$ be two languages and let $\diamond$ be a binary word operation. If the equation $X \diamond L = R$ (respectively, $L \diamond Y = R$) has a solution then the language $X_{\max} = (R^c \diamond^l L)^c$ (respectively, $Y_{\max} = (L \diamond^r R^c)^c$) is also a solution, namely one that includes all the other solutions to the equation.*

For example consider the case of scattered deletion and shuffle. The fact that the left inverse of scattered deletion is shuffle and viceversa helps us solve equations of the type

$$X \rightsquigarrow L = R, \quad X ⧢ L = R.$$

By Theorem 2.3, the maximal solutions to these equations, if they exist, are $X_{\max} = (R^c ⧢ L)^c$, respectively, $X_{\max} = (R^c \rightsquigarrow L)^c$. As REG is closed under scattered deletion [7] and shuffle [13], these maximal solutions are regular and can be effectively constructed in case R is regular. Note that the same languages are also solutions to the inequations $X \rightsquigarrow L \subseteq R$ and $X ⧢ L \subseteq R$, respectively, as a consequence of the following lemma, which can be shown using the same arguments as in the proof of Theorem 2.3.

Lemma 2.4. *If S is a solution to $X \diamond L \subseteq R$ (respectively, $L \diamond Y \subseteq R$) then also $(R^c \diamond^l L)^c$ (respectively, $(L \diamond^r R^c)^c$) is a solution, which includes S.*

2.4. Channels and error-detection

We recall the concepts of channel and error-detection from [11]. A channel is a binary relation γ that is domain preserving, that is, $\gamma \subseteq \Sigma^* \times \Sigma^*$ and (u, u) is in γ for all $u \in \mathrm{dom}\,(\gamma)$. The fact that (u, v) is in γ means that the word v can be received when u is transmitted via the channel γ. If, moreover, $u \neq v$ we say that v can be received from u with errors. The requirement that γ is domain preserving ensures that error-free communication via γ is possible. A channel γ is called rational if the relation γ is rational.

A language L is error-detecting for γ if no word in L_λ can be received from a different word in L_λ via γ. More formally, a language L is *error-detecting for a channel* γ iff for all words u and v in L_λ, if $(u, v) \in \gamma$ then $u = v$.

Remark 2.5. A language is error-detecting for γ if and only if it is error-detecting for γ^{-1}.

Next we list a few channels involving substitution, insertion, and deletion (SID) errors—see [12] for additional channels of this kind. We note that the subscript ‘s’ indicates scattered errors as opposed to burst errors [12].

$\delta_{\mathrm{s}}(m, \infty)$: consists of all pairs (u, v) such that v is obtained by deleting up to m symbols from u.

$\iota_{\mathrm{s}}(m, \infty)$: consists of all pairs (u, v) such that v is obtained by inserting up to m symbols in u.

$\sigma_{\mathrm{s}}(m, \infty)$: consists of all pairs (u, v) such that v is obtained by substituting up to m symbols of u with different alphabet symbols. Equivalently, this channel consists of all pairs (u, v) such that $|u| = |v|$ and the Hamming distance $H(u, v)$ is at most m.

$(\sigma \odot \delta)_{\mathrm{s}}(m, \infty)$: consists of all pairs (u, v) such that v is obtained by performing a total of up to m substitutions and deletions in u.

$(\sigma \odot \iota)_{\mathrm{s}}(m, \infty)$: consists of all pairs (u, v) such that v is obtained by performing a total of up to m substitutions and insertions in u.

3. Code-related properties as solutions to language inequations

A language K is said to be a (*uniquely decodable*) *code*, if every word w in K^* has a unique factorization over K, that is, there is only one sequence of words $w_1, \dots, w_n$ in K, for some $n \geqslant 0$, such that $w = w_1 \cdots w_n$. A language property, say $\mathcal{P}$, can be viewed as the set of all languages having that property. Using this interpretation, many natural code-related properties can be viewed as solution sets to language inequations involving binary word operations. We provide in the following several examples. The reader is referred to [15] or [5], for instance, for details on codes.

Example 3.1. A language K is a prefix (respectively, suffix) code if $ux \in K$ (respectively, $xu \in K$) implies $x = \lambda$, for all words $u \in K$ and $x \in \Sigma^*$. Let $\mathcal{P}$ be the "prefix-code" property. Then $\mathcal{P}$ is the solution set of $(X \longrightarrow_{rq} \Sigma^+) \subseteq X^c$ with the constraint $X \subseteq \Sigma^+$. Similarly the "suffix-code" property is the solution set of $(X \longrightarrow_{lq} \Sigma^+) \subseteq X^c$ with $X \subseteq \Sigma^+$.

Example 3.2. A language K is an infix code if $xuy \in K$ implies $x = y = \lambda$, for all words $u \in K$ and $x, y \in \Sigma^*$. It is an outfix code if $u_1u_2 \in K$ and $u_1xu_2 \in K$ implies $x = \lambda$, for all words $u_1, u_2, x \in \Sigma^*$. Let $\mathcal{P}$ be the "infix-code" property. Then $\mathcal{P}$ is the solution set of $(X \rightleftharpoons \Sigma^+) \subseteq X^c$ with $X \subseteq \Sigma^+$. Similarly, the "outfix-code" property is the solution set of $(X \longrightarrow \Sigma^+) \subseteq X^c$ with $X \subseteq \Sigma^+$.

Example 3.3. A language K is a hypercode if $u \in v \amalg \Sigma^*$ implies $u = v$, for all words $u, v \in K$. The "hypercode" property is exactly the solution set of $(X \amalg \Sigma^+) \subseteq X^c$ with $X \subseteq \Sigma^+$.

The next examples show how certain "error-detection" properties can also be modelled in terms of solution sets to language equations. Let γ be a channel. We write $\mathcal{P}_\gamma$ for the "γ-error-detecting language" property, that is $\mathcal{P}_\gamma$ is the class of all languages that are error-detecting for γ.

Example 3.4. Let γ be the channel $\delta_{\mathrm{s}}(m, \infty)$, i.e., $(u, v) \in \gamma$ iff v is obtained from u by at most m deletions. Then $\mathcal{P}_\gamma$, the set of all languages which are error-detecting for γ, is exactly the set of solutions of $X_\lambda \rightsquigarrow (\Sigma \bigcup \dots \bigcup \Sigma^m) \subseteq X_\lambda^c$. Indeed, let $X \in \mathcal{P}_\gamma$. Consider $z \in x \rightsquigarrow y$ with $x \in X_\lambda$ and $y \in \Sigma^+$ with $|y| \leqslant m$. We want to show $z \notin X_\lambda$. As z is obtained from x using at least 1 and at most m scattered deletions, it follows that (x, z) is in γ and $x \neq z$. Hence $z \notin X_\lambda$. Conversely, suppose X satisfies the inequation but $X \notin \mathcal{P}_\gamma$. Then there are two different words x and z in X_λ such that $(x, z) \in \gamma$. This implies $z \in X_\lambda \rightsquigarrow (\Sigma \bigcup \dots \bigcup \Sigma^m)$ and, therefore, $z \in X_\lambda^c$—a contradiction. Hence, $X \in \mathcal{P}_\gamma$.

Example 3.5. Let γ be an insertion channel $\gamma = \iota_{\mathrm{s}}(m, \infty)$, i.e., $(u, v) \in \gamma$ iff v is obtained from u by at most m insertions. We have that $\mathcal{P}_\gamma$, the set of all languages which are error-detecting for γ, is exactly the set of solutions of $X \amalg (\Sigma \bigcup \dots \bigcup \Sigma^m) \subseteq X^c$, or equivalently, the set of solutions of $X_\lambda \amalg (\Sigma \bigcup \dots \bigcup \Sigma^m) \subseteq X_\lambda^c$.

4. Using word operations to model channels

Let $\diamond$ be a binary word operation and L be a language. The pair $(\diamond, L)$ plays an important role in the sequel.

Definition 4.1. Let L be a language and let $\diamond$ be a binary word operation.
(i) The binary relation $[\diamond L]$ consists of all pairs (u, v) of words such that $v \in u\diamond L$.
(ii) The operation $\diamond$ is called L-rational if $[\diamond L]$ is a rational relation.

Recall that, for a binary operation $\gamma \subseteq \Sigma^* \times \Sigma^*$ and a word $u \in \Sigma^*$, we defined $\gamma(u) = \{v \mid (u, v) \in \gamma\}$.

Lemma 4.2. (i) *For every binary operation $\diamond$ and languages K and L, one has that $[\diamond L](K) = K\diamond L$.*
(ii) *For every binary operation $\diamond$ and language L, $[\diamond^l L] = [\diamond L]^{-1}$.*
(iii) *For every binary relation γ, there is a binary operation $\diamond$ and a language L such that $\gamma = [\diamond L]$.*

Proof. (i) Follows easily from the above definition.
(ii) We have $(u, v) \in [\diamond^l L]$ iff $v \in u\diamond^l L$ iff $u \in v\diamond L$ iff $(v, u) \in [\diamond L]$ iff $(u, v) \in [\diamond L]^{-1}$.
(iii) There are many ways to define $\diamond$ and L from γ. For example, consider the relation $B = \{(v, u, z) : z \in \Sigma^*$ and $(u, v) \in \gamma\}$. Then $\gamma = [\diamond\Sigma^*]$, where $\diamond$ is the binary operation whose characteristic relation is B. □

From the examples in Section 3 we understand that there is a close connection between channels and pairs of the form $(\diamond, L)$. For example, the channel $\delta_s(m, \infty)$ is equal to $[\rightsquigarrow(\Sigma^0 \cup \cdots \cup \Sigma^m)]$ and the channel $\iota_s(m, \infty)$ is equal to $[\amalg(\Sigma^0 \cup \cdots \cup \Sigma^m)]$. As $\amalg$ is the left inverse of $\rightsquigarrow$, the above lemma implies that the channel $\delta_s(m, \infty)$ is the inverse of the channel $\iota_s(m, \infty)$. By Remark 2.5, this in turn implies that a language is error-detecting for $\delta_s(m, \infty)$ if and only if it is error-detecting for $\iota_s(m, \infty)$.
Next we consider two natural binary word operations related to channels with substitution errors.

Definition 4.3. If $u, v \in \Sigma^*$ then we define the *substitution in u by v* as $u\bowtie v = \{u_1v_1u_2v_2 \ldots u_kv_ku_{k+1} \mid k\geqslant 0, u = u_1a_1u_2a_2 \ldots u_ka_ku_{k+1}, v = v_1v_2 \ldots v_k, a_i, v_i \in \Sigma, 1\leqslant i\leqslant k, a_i \neq v_i, \forall i, 1\leqslant i\leqslant k\}$.

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

Example 4.4. Let $\gamma = \sigma_s(m, \infty)$. Then $\mathcal{P}_\gamma$ is the solution set of the inequation $X\bowtie(\Sigma \bigcup \ldots \bigcup \Sigma^m) \subseteq X^c$. Moreover, the channel $\sigma_s(m, \infty)$ is equal to $[\bowtie(\Sigma^0 \cup \cdots \cup \Sigma^m)]$.

Definition 4.5. If $u, v \in \Sigma^*$ then we define the *substitution in u of v* as $u\triangle v = \{u_1a_1u_2a_2 \ldots u_ka_ku_{k+1} \mid k\geqslant 0, u = u_1v_1u_2v_2 \ldots u_kv_ku_{k+1}, v = v_1v_2 \ldots v_k, a_i, v_i \in \Sigma, 1\leqslant i\leqslant k, a_i \neq v_i, \forall i, 1\leqslant i\leqslant k\}$.

Lemma 4.6. *The operation $\bowtie$ is the left-inverse of $\triangle$.*

Proof. Let $w \in u\bowtie v$. Then $u = u_1a_1u_2a_2 \ldots u_ka_ku_{k+1}$, $v = v_1v_2 \ldots v_k$ and $w = u_1v_1u_2v_2 \ldots u_kv_ku_{k+1}$ for some $u_i \in \Sigma^*$, $a_i, v_i \in \Sigma$, $a_i \neq v_i$, $1\leqslant i\leqslant k$. This means $u \in w\triangle v$.
Conversely, let $u \in (w\triangle v)$. Then $w = w_1v_1w_2v_2 \cdots w_kv_kw_{k+1}$, $v = v_1v_2 \ldots v_k$ and u is equal to $w_1a_1w_2a_2 \cdots w_ka_kw_{k+1}$ for some $w_i \in \Sigma^*$, $a_i, v_i \in \Sigma$, $a_i \neq v_i$, $1\leqslant i\leqslant k$. This means $w \in (u\bowtie v)$. □

By Theorem 2.3 the equations $X\bowtie L = R$, $X\triangle L = R$ have as maximal solutions (if any) $X_{\max} = (R^c\triangle L)^c$, respectively, $Y_{\max} = (R^c\bowtie L)^c$.

The operations $\triangle$ and $\bowtie$ have a closer relation when the right operand is a *length-closed* language. A language L is length-closed if, for every $n \geqslant 0$, when a word of length n is in L then all words of length n are in L. An example of such a language is $\Sigma^0 \cup \cdots \cup \Sigma^m$.

Remark 4.7. For every length-closed language L, $[\triangle L] = [\bowtie L]$. Therefore, $\sigma_s(m, \infty) = [\triangle(\Sigma^0 \cup \cdots \cup \Sigma^m)]$.

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

Definition 4.8. For any words $u, v \in \Sigma^*$ of the same length and with Hamming distance $H(u, v) = k$, for some nonnegative integer k, $u \triangleright v$ is the set of words

$$b_1 b_2 \ldots b_k, \ b_i \in \Sigma, 1 \leqslant i \leqslant k,$$

such that $u = u_1 a_1 \cdots u_k a_k u_{k+1}$, $v = u_1 b_1 \cdots u_k b_k u_{k+1}$ and, for all i, $1 \leqslant i \leqslant k$, $a_i \neq b_i$.

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

Example 4.9. If $L_1 = \{a^n b^n | n \geqslant 1\}$ and $L_2 = \{b^m | m \geqslant 1\}$, then $L_1 \triangleright L_2 = b^*$. (We can only perform $a^n b^n \triangleright b^{2n}$ which gives b^n.) On the other hand, $L_2 \triangleright L_1 = a^*$. Hence, the operation $\triangleright$ is not commutative.

Example 4.10. In general, if $L \subseteq \Sigma^*$ and $a \in \Sigma$ then $L \triangleright a^* \subseteq a^*$, $L \triangleright a^* = \{a^{|w| - |w|_a} \mid w \in L\}$, where $|w|_a$ is the number of a's occurring in the word w.

Note that $\triangleright$ is the right inverse of $\bowtie$, and the reversed $\triangleright$ is the right inverse of $\triangle$. Consequently, by Theorem 2.3, the solutions to the equations $L \bowtie Y = R$ and $L \triangleright Y = R$ (if any) are $Y_{\max} = (L \triangleright R^c)^c$, respectively, $Y_{\max} = (R^c \triangleright L)^c$.

To model more complex channels we need the concept of composition of two word operations. We shall assume that the symbol ‘;’ is not in the alphabet Σ.

Definition 4.11. Given two binary operations $\diamondsuit_1$ and $\diamondsuit_2$, define the binary operations $(\diamondsuit_1 \diamondsuit_2)$ and $(\diamondsuit_1; \diamondsuit_2)$ as follows:

For all words $u, w, v \in \Sigma^*$ $w \in u(\diamondsuit_1 \diamondsuit_2)v$ if and only if $w \in (u \diamondsuit_1 v_1) \diamondsuit_2 v_2$ for some words v_1 and v_2 with $v = v_1 v_2$. For all words $u, w \in \Sigma^*$ and $v \in \Sigma^*; \Sigma^*$, $w \in u(\diamondsuit_1; \diamondsuit_2)v$ if and only if $w \in (u \diamondsuit_1 v_1) \diamondsuit_2 v_2$ for some words v_1 and v_2 with $v = v_1; v_2$.

Next we provide some observations concerning the two composition operations.

A language L is *commutative*, if $xy \in L \Leftrightarrow yx \in L$ for all words x and y.

Proposition 4.12. *Let L, L_1, L_2 be languages over Σ. The following statements hold true.*

(i) $(L \diamondsuit_1 L_1) \diamondsuit_2 L_2 = L(\diamondsuit_1; \diamondsuit_2) L_1; L_2$.

(ii) $[(\Diamond_1;\Diamond_2)L_1;L_2] = [\Diamond_2 L_2]\circ[\Diamond_1 L_1]$.
(iii) $[(\Diamond_1;\Diamond_2)^l L_1;L_2] = [(\Diamond_2^l;\Diamond_1^l)L_2;L_1]$.
(iv) *If L is commutative then* $[(\Diamond_1\Diamond_2)^l L] = [(\Diamond_2^l\Diamond_1^l)L]$.

Proof. (i) Follows easily from the definition of composition.

(ii) We have that $(u,v)\in[(\Diamond_1;\Diamond_2)L_1;L_2]$ iff $v\in(u\Diamond_1 L_1)\Diamond_2 L_2$ iff there is a word z such that $z\in u\Diamond_1 L_1$ and $v\in z\Diamond_2 L_2$ iff there is a word z such that $(u,z)\in[\Diamond_1 L_1]$ and $(z,v)\in[\Diamond_2 L_2]$ iff $(u,v)\in[\Diamond_2 L_2]\circ[\Diamond_1 L_1]$.

(iii) We use part (i) and Lemma 4.2: $[(\Diamond_1;\Diamond_2)^l L_1;L_2] = [(\Diamond_1;\Diamond_2)L_1;L_2]^{-1} = [\Diamond_1 L_1]^{-1}\circ[\Diamond_2 L_2]^{-1} = [\Diamond_1^l L_1]\circ[\Diamond_2^l L_2] = [(\Diamond_2^l;\Diamond_1^l)L_2;L_1]$

(iv) Similar to the above. □

One can verify that the channel $(\sigma\odot\delta)_{\mathrm{s}}(m,\infty)$ is equal to $[(\rightsquigarrow\bowtie)(\Sigma^0\cup\cdots\cup\Sigma^m)]$, and the channel $(\sigma\odot\iota)_{\mathrm{s}}(m,\infty)$ is equal to $[(\triangle\amalg)(\Sigma^0\cup\cdots\cup\Sigma^m)]$. Hence, the following result holds.

Corollary 4.13. *The inverse of the channel $(\sigma\odot\delta)_{\mathrm{s}}(m,\infty)$ is $(\sigma\odot\iota)_{\mathrm{s}}(m,\infty)$; therefore, a language is error-detecting for $(\sigma\odot\delta)_{\mathrm{s}}(m,\infty)$ if and only if it is error-detecting for $(\sigma\odot\iota)_{\mathrm{s}}(m,\infty)$.*

We note that analogous results for the property of error-correction have been obtained in [10] using different tools. Now let $\gamma_1 = \sigma_{\mathrm{s}}(m_1,\infty)\oplus\delta_{\mathrm{s}}(m_2,\infty)$ be the channel consisting of all pairs (u,v) such that v is obtained from u using at most m_2 deletions and at most m_1 substitutions. Let $\gamma_2 = \sigma_{\mathrm{s}}(m_1,\infty)\oplus\iota_{\mathrm{s}}(m_2,\infty)$ be the channel consisting of all pairs (u,v) such that v is obtained from u using at most m_2 insertions and at most m_1 substitutions. Then, it follows that

$$\gamma_1 = [(\rightsquigarrow;\bowtie)(\Sigma^0\cup\cdots\cup\Sigma^{m_2});(\Sigma^0\cup\cdots\cup\Sigma^{m_1})],$$

$$\gamma_2 = [(\triangle;\amalg)(\Sigma^0\cup\cdots\cup\Sigma^{m_1});(\Sigma^0\cup\cdots\cup\Sigma^{m_2})].$$

Corollary 4.14. *The inverse of the channel $\sigma_{\mathrm{s}}(m_1,\infty)\oplus\delta_{\mathrm{s}}(m_2,\infty)$ is the channel $\sigma_{\mathrm{s}}(m_1,\infty)\oplus\iota_{\mathrm{s}}(m_2,\infty)$; therefore, a language is error-detecting for the channel $\sigma_{\mathrm{s}}(m_1,\infty)\oplus\delta_{\mathrm{s}}(m_2,\infty)$ if and only if it is error-detecting for $\sigma_{\mathrm{s}}(m_1,\infty)\oplus\iota_{\mathrm{s}}(m_2,\infty)$.*

5. Closure properties of 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 this section we investigate such closure properties for the scattered substitution operations, namely $\bowtie$, $\triangle$, $\triangleright$.

Proposition 5.1. *If L and R are languages over the alphabet Σ, R a regular one, $L\triangle R$ is the image of L through a λ-free gsm. Moreover, the gsm realizes the relation $[\triangle R]$.*

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

$$P' = \{sa \to as | s \in S, a \in \Sigma\} \cup \{sa \to bs' | sa \to s' \in P, a \neq b\}.$$

It is clear that $g(u_1 v_1 \cdots u_k v_k u_{k+1}) = \{u_1 a_1 \cdots u_k a_k u_{k+1} \mid v = v_1 \cdots v_k \in R \text{ and } a_i \neq v_i\}$ and therefore $g(L) = L \triangle R$ for any language $L \subseteq \Sigma^+$. Moreover, it follows that (u, u') is in $[\triangle R]$ if and only if $u' \in g(u)$, for all words u and u'. □

Corollary 5.2. *REG and CF are closed under $\triangle$ with regular languages.*

Proposition 5.3. *There exist two linear languages L_1, L_2 such that $L_1 \triangle L_2$ is not context-free.*

Proof. Let $\Sigma = \{a, b, c, d, f, \$\}$ and consider the two context-free languages over Σ

$$L_1 = \{a^n (bc)^n \$ (df)^m | n, m \geqslant 1\},$$

$$L_2 = \{c^n d^n | n \geqslant 1\}.$$

Then

$$(L_1 \triangle L_2) \cap a^* b^* \$ f^* = \{a^n b^{2n} \$ f^{2n} | n \geqslant 1\}$$

As CF is closed under intersection with regular languages it follows that $L_1 \triangle L_2$ is not a context-free language. □

Corollary 5.4. *CF is not closed under $\triangle$.*

Proposition 5.5. *CS is closed under $\triangle$.*

Proof. Let L_1, L_2 be two context-sensitive languages over Σ and let $\Sigma' = \{a' |\ a \in \Sigma\}$, $\Sigma'' = \{a'' |\ a \in \Sigma\}$. Consider the gsm $g = (S, \Sigma, \Sigma \cup \Sigma', s_0, F, P)$, with $S = \{s_0\} = F$, $P = \{s_0 a \to a s_0, s_0 a \to a' s_0 | a \in \Sigma\}$, that transforms some letters in their primed versions. Consider now the morphisms $h : \Sigma \to \Sigma''$, $h(a) = a''$, $a \in \Sigma$, and $h' : \Sigma \cup \Sigma' \cup \Sigma''$, $h'(a) = a$, $h'(a') = a'$, $h'(a'') = \lambda$.

We claim that $L_1 \triangle L_2 = g'\{h'[[g(L_1) ⧢ h(L_2)] \cap [\bigcup_{a \in \Sigma} \Sigma^* a' a'' \Sigma^*]^*]\}$ where g' is the gsm $g' = (S', \Sigma \cup \Sigma', \Sigma, s', F', P')$ and $S' = \{s'\} = F'$, $P' = \{s'a \to as' | a \in \Sigma\} \cup \{s'a' \to bs' | a \neq b, a, b \in \Sigma\}$.

Indeed, given a word $u = u_1 v_1 u_2 v_2 \ldots u_k v_k u_{k+1} \in L_1$ and $v = v_1 v_2 \ldots v_k \in L_2$, $v_i \in \Sigma$, $u_i \in \Sigma^*$, $1 \leqslant i \leqslant k$,

$$[g(u) ⧢ h(v)] \cap [\bigcup_{a \in \Sigma} \Sigma^* a' a'' \Sigma^*]^*$$

produces $u_1 v_1' v_1'' u_2 v_2' v_2'' \ldots u_k v_k' v_k'' u_{k+1}$.

The intersection with $(\bigcup_{a \in \Sigma} \Sigma^* a' a'' \Sigma^*)^*$ ensures that only words $g(u)$ and $h(v)$, where v is a subword of u, are shuffled, and only words where a primed letter is followed by an identical double primed letter are kept. Applying h' to $u_1 v_1' v_1'' u_2 v_2' v_2'' \dots u_k v_k' v_k'' u_{k+1}$ erases the double primed letters producing $u_1 v_1' u_2 v_2' \dots u_k v_k' u_{k+1}$, while g' replaces every primed letter with a different one, resulting in $u_1 a_1 u_2 a_2 \dots u_k a_k u_{k+1} \in u \triangle v$, as $a_i \neq v_i$, $1 \leqslant i \leqslant k$.

A morphism h is termed a k-linear erasing with respect to L iff, for each $w \in L$, $|w| \leqslant k|h(w)|$. Note that h' is a 2-linear erasing with respect to the language it is applied to, as it erases at most half of each word. The proposition now follows as CS is closed under k-linear erasing as well as all the other operators involved. □

Proposition 5.6. *If $L_1, L_2 \subseteq \Sigma^*$, L_2 regular, then $L_1 \bowtie L_2$ is the image of L_1 through a λ-free gsm. Moreover, the gsm realizes the relation $[\bowtie L_2]$.*

Proof. We have that $L_1 \bowtie L_2 = [\bowtie L_2](L_1) = [\triangle^l L_2](L_1) = [\triangle L_2]^{-1}(L_1) = g^{-1}(L_1)$, where g is the gsm realizing $[\triangle L_2]$—see Proposition 5.1. The claim follows when we recall that g^{-1} is obtained from g by simply replacing every production $sa \to bt$ of g with the production $sb \to at$ [17]. □

Corollary 5.7. *REG, CF and CS are closed under $\bowtie$ with regular languages.*

Proposition 5.8. *CF is not closed under $\bowtie$.*

Proof. Use exactly the same languages L_1 and L_2 as in Proposition 5.3 for the operation $\triangle$ and the language

$$(L_1 \bowtie L_2) \cap a^* c^* \$ d^* = \{a^n c^{2n} \$ d^{2n} \mid n \geq 1\}. \qquad \square$$

Proposition 5.9. *CS is closed under $\bowtie$.*

Proof. Let L_1, L_2 be two context-sensitive languages over Σ and let $\Sigma' = \{a' | a \in \Sigma\}$, $\Sigma'' = \{a'' | a \in \Sigma\}$.

Construct the gsm $g = (S, \Sigma, \Sigma \cup \Sigma', s_0, F, P)$ where $S = \{s_0\}$, $F = \{s_0\}$, $P = \{sa \to as | a \in \Sigma\} \cup \{sa \to a's | a \in \Sigma\}$. The nonerasing gsm g nondeterministically changes some letters into their primed versions.

Consider now the morphism $h : \Sigma \to \Sigma''$, $h(a) = a''$, $a \in \Sigma$, and the morphism $h' : \Sigma \cup \Sigma' \cup \Sigma'' \to \Sigma$ defined as $h'(a) = a$, $h'(a') = \lambda$, $h'(a'') = a$, $a \in \Sigma$.

We claim that $L_1 \bowtie L_2 = h'[[g(L_1) ⧢ h(L_2)] \cap [\bigcup_{a,b \in \Sigma, a \neq b} (\Sigma^* a' b'' \Sigma^*)]^*]$.

Indeed, let us consider a word $u = u_1 a_1 u_2 a_2 \dots u_k a_k u_{k+1} \in L_1$ and $v = v_1 v_2 \dots v_k \in L_2$, $a_i \neq v_i$, $1 \leqslant i \leqslant k$.

We have that

$$(g(u) ⧢ h(v)) \cap [\bigcup_{a,b \in \Sigma, a \neq b} \Sigma^* a' b'' \Sigma]^*$$

produces words of the form $u_1 a_1' v_1'' u_2 a_2' v_2'' \dots u_k a_k' v_k'' u_{k+1}$.

The intersection with $[\bigcup_{a,b\in\Sigma,a\neq b}\Sigma^*a'b''\Sigma^*]^*$ ensures that only those words $g(u)$ and $h(v)$ are shuffled where each letter of v is different from a letter in u, and only those words are kept from the shuffle in which the letters in $h(u)$ and their "different" counterparts are adjacent. The morphism h' afterwards erases all the primed letters and transforms the double primed letters into ordinary ones, resulting in $u_1v_1u_2v_2\dots u_kv_ku_{k+1}\in u\bowtie v$.

Note that h' is a 2-linear erasing 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 $\bowtie$. □

Proposition 5.10. *If $L_1, L_2 \subseteq \Sigma^*$, L_2 regular, then there exists a gsm g with erasing such that $g(L_1) = L_1 \triangleright L_2$. Moreover, the gsm realizes the relation $[\triangleright L_2]$.*

Proof. Let L_2 be a regular language, $A = (S, \Sigma, s_0, F, P)$ be a finite automaton, $L(A) = L_2$. Construct the gsm $g = (S, \Sigma, \Sigma, s_0, F, P)$ where $P' = \{sa \to s'|sa \to s' \in P\} \cup \{sa \to bs'|sb \to s' \in P, b \neq a\}$.

Then $g(L_1) = L_1 \triangleright L_2$. Indeed, consider $u = u_1a_1\dots u_ka_ku_{k+1} \in L_1$, $v = u_1b_1u_2b_2\dots u_kb_ku_{k+1}$, $a_i \neq b_i$, $1 \leqslant i \leqslant k$.

The gsm g applied to u works as follows. Rules of the type $sa \to s' \in P$ erase subwords u_i that are common between u and v. Rules $sa \to bs'$ where $sb \to s' \in P$, $b \neq a$ read the letters a where words u and v differ and replace them with the corresponding letters in v.

The fact that the set of final states is F, the set of final states of A, ensures that only words $w \in u \triangleright v$, $v \in L_2$ reach a final state. Moreover, it is evident that g realizes the relation $[\triangleright L_2]$. □

Corollary 5.11. *CF, REG are closed under $\triangleright$ with regular languages.*

Proof. It follows as REG, CF are closed under gsm mappings. □

Proposition 5.12. *CS is not closed under $\triangleright$ with regular languages.*

Proof. Let L be a recursively enumerable language over Σ and let a, b be different symbols not in Σ. Then, [14, p. 89] there exists a CS language L_1 such that (i) L_1 consists of words of the form a^ibw, $i \geqslant 0$, $w \in L$, and (ii) for every $w \in L$, there is an $i \geqslant 0$ such that $a^ibw \in L_1$.

Let $\Sigma' = \{c'|\ c \in \Sigma\}$ and $\Sigma'' = \{c''|\ c \in \Sigma\}$. Consider now the morphism h on $\Sigma \cup \{a, b\}$ defined by $h(a) = a$, $h(b) = b$, $h(c) = cc'$ for all $c \in \Sigma$.

We claim that $h_1(L) = K$, where

$$K = [h(L_1) \triangleright a^*b(\bigcup_{c\in\Sigma} cc'')^*] \cap (\Sigma'')^*$$

and $h_1 : \Sigma \longrightarrow \Sigma''$ is the morphism defined as $h_1(c) = c''$ for all $c \in \Sigma$. We leave it to the reader to verify that every word in $h_1(L)$ also belongs to K. Now take a word $w \in K$. Then there exist words $u \in h(L_1)$,

$v \in a^*b(\bigcup_{c\in\Sigma} cc'')^*$ such that $w \in u \triangleright v$. The words u, v are of the form $u = a^i ba_1a_1'a_2a_2' \ldots a_ka_k'$ respectively $v = a^j bb_1b_1''b_2b_2'' \ldots b_mb_m''$ for some $i, j, m, k \geqslant 0$.

If $i \neq j$ then w would contain letters a or b which contradicts $w \in (\Sigma'')^*$. Consequently, $i = j$. As $|u| = |v|$, it follows that $m = k$.

If there would exist $1 \leqslant l \leqslant k$ with $a_l \neq b_l$ then the word w would contain the letter $b_l \in \Sigma$ which contradicts $w \in (\Sigma'')^*$. Consequently, for all $1 \leqslant l \leqslant k$, $a_l = b_l$. We can easily see now that, following these considerations, $w = b_1''b_2'' \ldots b_k''$ with $b_1b_2 \ldots b_k \in L$, and the claim follows.

It follows then that the class CS is not closed under $\triangleright$ with regular languages, as this class is closed under nonerasing morphisms, intersection with regular languages and, if L is a noncontext-sensitive language $h_1(L)$ will have the same property. (If $h_1(L)$ were context-sensitive then L, which equals the image of $h_1(L)$ through a nonerasing morphism that transforms all double primed letters into normal ones, would also be context-sensitive.) □

Proposition 5.13. *The family CF is not closed under* $\triangleright$.

Proof. Let $\Sigma = \{a, b, c, e, f, g, x, y, z\}$ and consider the languages

$$L_1 = \{(ax)^i(by)^i(cz)^k \mid i, k \geqslant 0\}, \quad L_2 = \{(ex)^l(fy)^m(gz)^m \mid l, m \geqslant 0\}.$$

Then $[(ax)^i(by)^i(cz)^k \triangleright (ex)^l(fy)^m(gz)^m] \cap e^*f^*g^* = \{e^if^ig^i \mid i \geqslant 0\}$ which is not context-free. □

6. Error-detection and the inequation $X \diamond L \subseteq X^c$ with $X \subseteq M$

The examples provided in Sections 3 and 4 reveal the following pattern: many natural code-related properties can be reduced to the property of error-detection by varying the channel involved. At the same time, for many channels the property of error-detection can be studied via the inequation

$$X \diamond L \subseteq X^c \text{ with } X \subseteq M \qquad (*)$$

by varying the operator $\diamond$ and the languages L and M. More specifically, consider the case where the pair $(\diamond, L)$ satisfies the condition

For all $u \in \Sigma^*$, $u \notin u\diamond L$ and $u \in u \diamond \lambda$ $\qquad$ $\mathrm{C}(\diamond, L)$.

When condition $\mathrm{C}(\diamond, L)$ is satisfied, the relation $[\diamond L_\lambda]$ is a channel and it follows that a language is error-detecting for $[\diamond L_\lambda]$ if and only if it is a solution of $(*)$ with $M = \Sigma^*$. With this interpretation of the inequation $(*)$, we have that a language is a prefix code (respectively, suffix, infix, outfix, hypercode) if and only if it is error-detecting for the channel $[\longrightarrow_{rq} \Sigma^*]$ (respectively,$[\longrightarrow'_{lq} \Sigma^*]$, $[\rightleftharpoons \Sigma^*]$, $[\longrightarrow \Sigma^*]$, $[\amalg \Sigma^*]$).

Definition 6.1. Inequation $(*)$ is of *type* (c), if condition $\mathrm{C}(\diamond, L)$ is satisfied.

In this section, we provide some observations and obtain general statements about the solutions of the inequation $(*)$ which are meaningful to error-detection, hence also to the code properties reducible to the error-detection property. In particular, in Corollary 6.7 we obtain a characterization of the maximal solutions of a type (c) inequation, which yields a method for deciding whether a given regular solution is maximal—see Proposition 6.14 and the discussion following that proposition. A consequence of this result is that one can use the same method to decide whether a given regular prefix code, or suffix code, or infix code, or error-detecting language is maximal because each of these code properties is definable via an inequation of type (c), as shown in Section 3.

An important concept in our considerations is the residue of a solution.

Definition 6.2. Let S be a solution of $(*)$. The *residue* of S is the language $M - (S \cup S\diamondsuit L \cup S\diamondsuit^l L)$.

Proposition 6.3. (i) *If S is a solution of $(*)$ then every subset of S is also a solution of $(*)$.*

(ii) *Every solution of $(*)$ is included in a maximal solution of $(*)$.*

(iii) *If the equation $(*)$ is of type (c) then $\{w\}$ is a solution of $(*)$, for every word w in M.*

Proof. (i) Let S_1 be a subset of S and let w be a word in $S_1\diamondsuit L$. As $S_1\diamondsuit L$ is a subset of $S\diamondsuit L$, it follows that $w \in S^c$, hence also, $w \in S_1^c$.

(ii) Let $\mathcal{P}$ be the solution set of $(*)$ and let $S = \{S_i : i \in I\}$ be any totally ordered subset of $\mathcal{P}$. We show that the upper bound $\bigcup_{i\in I} S_i$ of S is a solution of $(*)$ as well; then the claim follows by Zorn's lemma. Let $z \in s\diamondsuit u$ for some $s \in \bigcup S_i$ and $u \in L$. Then $s \in S_j$, for some $j \in I$. If also $z \in \bigcup S_i$ then $z \in S_i$ for some $i \in I$. Let $k = \max\{i, j\}$. Then $z, x \in S_k$ and $(S_k\diamondsuit L) \bigcap S_k \neq \emptyset$, which contradicts the fact that S_k is a solution of $(*)$.

(iii) Obvious. □

The following is based on the proof of Theorem 2.3.

Lemma 6.4. *For any languages X, Y, Z and for any binary operator $\diamondsuit$,*

$$X\diamondsuit Y \subseteq Z \Leftrightarrow X \subseteq (Z^c\diamondsuit^l Y)^c \Leftrightarrow Y \subseteq (X\diamondsuit^r Z^c)^c.$$

Proof. We consider only the first equivalence: "$\Rightarrow$" Let $x \in X$, but suppose $x \in Z^c\diamondsuit^l Y$; then $x \in t\diamondsuit^l Y$ for some $t \in Z^c$, which implies $t \in x\diamondsuit Y$ and $t \in X\diamondsuit Y \subseteq Z$; a contradiction.

"$\Leftarrow$" Let $z \in x\diamondsuit Y$, for some $x \in X$, but suppose $z \in Z^c$. Then $x \in z\diamondsuit^l Y \subseteq Z^c\diamondsuit^l Y$. As $(Z^c\diamondsuit^l Y) \subseteq X^c$, $x \in X^c$; a contradiction. □

Corollary 6.5. (i) *Eq. $(*)$ is equivalent to*

$$X\diamondsuit^r X \subseteq L^c \ \mathit{with}\ X \subseteq M \tag{$**$}$$

which in turn is equivalent to

$$X\diamondsuit^r X \subseteq \mathrm{dom}_2(\diamondsuit) - L \ \ \mathit{with}\ \ X \subseteq M.$$

(ii) *A language is a solution of $(*)$ if and only if it is a solution of $X\diamondsuit^l L \subseteq X^c$ with $X \subseteq M$.*

Proof. $X\diamond L \subseteq X^c$ is equivalent to $L \subseteq (X\diamond^r X)^c$ which is equivalent to $X\diamond^r X \subseteq L^c$. As $\mathrm{im}\,(\diamond^r) = \mathrm{dom}_2(\diamond)$, $X\diamond^r X \subseteq \mathrm{dom}_2(\diamond)$ and the claim follows. The second part can be shown analogously. □

Proposition 6.6. *Let S be a solution of* $(*)$.
(i) *If the residue of S is empty, then S is a maximal solution of* $(*)$.
(ii) *If* $(*)$ *is of type* (c) *and the solution S is maximal, then the residue of S is empty*.
(iii) *If* $(*)$ *is of type* (c), *then* $S \cup \{w\}$ *is a solution of* $(*)$ *for every word w in the residue of S*.

Proof. (i) Assume $M \subseteq S \cup S\diamond L \cup S\diamond^l L$, but suppose there is $w \in M - S$ such that $T = S \cup \{w\}$ is a solution of $(*)$. As *w* is not in *S*, at least one of the following holds.

(a) *w* is in $S\diamond L$. In this case, $w \in z\diamond L$ for some $z \in S$, which implies $z \in w\diamond^l L \Rightarrow z \in T\diamond^l L \Rightarrow z \in T^c \Rightarrow z \notin S$, a contradiction.

(b) *w* is in $S\diamond^l L$. In this case, $w \in z\diamond^l L$ for some $z \in S$, which implies $z \in T\diamond L \Rightarrow z \in T^c \Rightarrow z \notin S$, a contradiction.

Hence, *S* must be maximal.

(ii) This is a consequence of (iii), which we prove next.

(iii) Assume $(*)$ is of type (c) and consider any word $w \in M$ such that *w* is not in $S \cup S\diamond L \cup S\diamond^l L$. Let $T = S \cup \{w\}$. We show that $T\diamond L \subseteq T^c$. Let $z \in T\diamond L$. We consider two cases.

(a) $z \in S\diamond L$. As *S* is a solution of $(*)$, $z \notin S$. Also, if $z = w$ then $w \in S\diamond L$, which contradicts our choice of *w*. Hence, $z \notin S \cup \{w\}$.

(b) $z \in w\diamond L$. Then $w \in z\diamond^l L$. If $z \in S$ then $w \in S\diamond^l L$, which is impossible again. Hence, $z \notin S$. If $z = w$ then $w \in w\diamond L$, which contradicts condition (c). Hence, $z \neq w$. It follows again that $z \notin T$. □

Corollary 6.7. *Let S be a solution to an inequation of type* (c). *Then S is maximal if and only if the residue of S is empty*.

Corollary 6.8. *If S is a solution of the equation* $X\diamond L = M - X$, *then S is a maximal solution of* $(*)$.

Proposition 6.9. *If S is a solution of the inequation* $X\diamond X \subseteq R$ *with* $X \subseteq M$, *then also each of the languages*

$$M \cap (R^c\diamond^l S)^c \cap ((R^c\diamond^l S)^c\diamond^r R^c)^c$$

and

$$M \cap (S\diamond^r R^c)^c \cap (R^c\diamond^l (S\diamond^r R^c)^c)^c$$

is a solution, which includes S.

Proof. Assume *S* is a solution of the given inequation and let *P* be the language $(R^c\diamond^l S)^c$. As *S* is a solution of $X\diamond S \subseteq R$, one has that also *P* is a solution which includes *S*. Hence, $P\diamond S \subseteq R$. Now this implies that *S* is a solution of the inequation $P\diamond Y \subseteq R$; therefore, also the language $(P\diamond^r R^c)^c$, call it

Q, is a solution which includes S. Hence, $P\diamond Q \subseteq R$ and, as $(P \cap Q)\diamond(P \cap Q)$ is a subset of $P\diamond Q$, it follows that the language $P \cap Q$ satisfies the inequation $X\diamond X \subseteq R$. As every subset of $P \cap Q$ also satisfies this inequation, we have that $M \cap P \cap Q$ is a solution of $X\diamond X \subseteq R$ with $X \subseteq M$. The statement about the second language can be shown analogously. □

Definition 6.10. Let L be a language. The set of *left* (*respectively, right*) *quotients of L with respect to* $\diamond$ is the set of languages of the form $L\diamond^l W$ (respectively, $W\diamond^r L$), where W is a subset of Σ^*.

Using the fact that $A \cap B^c = A - B$, for any languages A and B, Proposition 6.9 implies the following, where an expression of the form $M - A - B$ is shorthand for $(M - A) - B$.

Corollary 6.11. *If S is a solution of the inequation $X\diamond X \subseteq R$ with $X \subseteq M$ then there is a left quotient P_l and a right quotient P_r of R^c with respect to $\diamond$ such that each of the languages $M - P_l - (P_l^c\diamond^r R^c)$ and $M - P_r - (R^c\diamond^l P_r^c)$ is also a solution which includes S.*

Using the above results and the fact that $\diamond^{rl} = \diamond^{l'}$, for all binary operations $\diamond$, also the following holds true.

Corollary 6.12. *Every maximal solution of $(*)$ is of the form $M - P_l - (P_l^c\diamond L)$ and of the form $M - P_r - (P_r^c\diamond^l L)$, where P_l and P_r are left and right, respectively, quotients of L with respect to $\diamond^r$.*

We are interested in algorithms whose input involves equations of the form $(*)$. More specifically, we shall assume that $(*)$ is such that $\diamond$ is L-rational and M is regular. In this case, the equation is given effectively by a finite transducer realizing $[\diamond L]$ and a finite automaton accepting M.

The following result provides a uniform polynomial time algorithm for deciding properties of regular languages that are definable via an equation of the form $(*)$. For the proof of this and other results involving constructions and sizes of automata and transducers, we shall use the following notation—see also [17,11].

Notation: Let A and B be two trim λ-NFAs and let T be a trim transducer (in standard form).

- $A \cap B$ is a trim λ-NFA of size $O(|A||B|)$ accepting the language $L(A) \cap L(B)$.
- $A \cup B$ is a trim λ-NFA of size $O(|A| + |B|)$ accepting the language $L(A) \cup L(B)$.
- If A and B are DFAs then $A \sqcup B$ is a trim DFA of size $O(|A||B|)$ accepting the language $L(A) \cup L(B)$.
- If A is a DFA then A^c is a trim DFA of size $O(|A|)$ accepting the language $L(A)^c$.
- A_T is a trim λ-NFA accepting the language $R(T)(L(A)) = \{z \in \Sigma^* \mid (w, z) \in R(T),\ w \in L(A)\}$.
- T^{-1} is a trim transducer of size $O(|T|)$ realizing the relation $R(T)^{-1}$.

Proposition 6.13. *The following problem is decidable in time*:

$$O(|A|^2|T| + |A||B|).$$

Input: *A trim λ-NFA A, a DFA B, and a trim transducer T (in standard form) realizing $[\diamond K]$, for some binary operation $\diamond$ and language K.*
Output: *Y/N depending on whether $L(A)$ is a solution of $X\diamond K \subseteq X^c$ with $X \subseteq L(B)$.*

Proof. Testing whether $L(A) \subseteq L(B)$ is equivalent to testing whether $L(A) \cap L(B)^c = \emptyset$. This is possible when we construct the automaton $A \cap B^c$ of size $O(|A||B|)$, and test whether there is a path from the start state to a final state, which takes time linear with respect to the graph of the automaton using depth first search, for instance.

Now consider the problem of deciding whether $L(A)\diamond K$ is a subset of $L(A)^c$. By Lemma 4.2, this is equivalent to testing whether $[\diamond K](L(A)) \subseteq L(A)^c$. As the relation $[\diamond K]$ is realized by T, one has that $[\diamond K](L(A)) = L(A_T)$. Hence, the problem is whether $L(A_T) \cap L(A)$ is empty. As before, one constructs the automaton $A_T \cap A$ of size $O(|A|^2|T|)$ and tests whether there is a path from the start state to a final state. □

The assumption that B is a DFA as opposed to a λ-NFA is essential as, otherwise, computing the complement of a λ-NFA requires to convert it to a DFA, which in general would be of exponential size. In practice, however, the automaton B and possibly the transducer T are fixed and, therefore, not part of the input. In such cases the algorithm would require time $O(|A|^2|T|)$, or simply $O(|A|^2)$ when T is fixed.

Proposition 6.14. *The following problem is computable*:
Input: *A λ-NFA A, a λ-NFA B, and a transducer T realizing $[\diamond K]$, for some binary operation $\diamond$ and language K, such that $L(A)$ is a solution of $X\diamond K \subseteq X^c$ with $X \subseteq L(B)$.*
Output: *A λ-NFA accepting the residue of $L(A)$.*

Proof. Consider the language

$$W = L(A) \cup L(A)\diamond K \cup L(A)\diamond^l K.$$

By Lemma 4.2, W is equal to $L(A) \cup [\diamond K](L(A)) \cup [\diamond K]^{-1}(L(A))$. As T realizes the relation $[\diamond K]$ and T^{-1} realizes the relation $[\diamond K]^{-1}$, the problem can be solved if we first construct the λ-NFA $C = A \cup A_T \cup A_{T^{-1}}$ accepting the language W, and then construct the λ-NFA $B \cap C^c$ accepting the language $L(B) - W$, which is equal to the residue of $L(A)$. □

A consequence of the above is that one can decide whether the given solution $L(A)$ is maximal by testing whether the residue of $L(A)$ is empty, provided the equation is of type (c)—see Corollary 6.7. Moreover, the examples in Section 3 imply that one can decide whether a given regular prefix code, or suffix code, or infix code, or error-detecting language is maximal.

In the proof of the preceding proposition, even if A is a DFA the automaton A_T, or $A_{T^{-1}}$, might be a λ-NFA. In this case, computing C^c would require to convert C to a DFA. Thus, the above algorithm might require exponentially many steps. On the other hand, one hopes that when the given transducer is of a certain particular type (or even fixed), the residue of a solution can be computed in polynomial time. This possibility is explored in the next section.

7. Special cases and applications

7.1. Languages with finitely many quotients

Recall that, by Corollary 6.5, the inequation $(X\diamond L) \subseteq X^c$, $(*)$, is equivalent to $(X\diamond^r X) \subseteq L^c$. We want therefore to be able to solve inequations of the form $X\diamond X \subseteq R$, for a given language $R \subseteq \Sigma^*$ and unknown X. In [9], it is shown that if both the sets of left and right quotients of R^c with respect to $\diamond$ are finite one can identify all the maximal solutions of the equation $X \diamond X = R$. The same argument can be applied also for solving the inequation $X \diamond X \subseteq R$. Here we improve this result by showing how to identify all the maximal solutions of our inequation when one of the quotient sets is known to be finite. Indeed, suppose that the set of left quotients of R^c with respect to $\diamond$ is finite: $P_1, \ldots, P_n$. According to Corollary 6.11, the following method would produce all the maximal solutions of the inequation $X \diamond X \subseteq R$ with $X \subseteq M$:

(i) For each $i = 1, \ldots, n$, let T be the language $P_i^c \cap (P_i^c \diamond^r R^c)^c$. If $T \diamond T$ is a subset of R then add $T \cap M$ in the list of solutions.

(ii) Remove from the list any solutions that are proper subsets of other solutions.

It should be clear that, if the set of right quotients of R^c is finite, then we can use a similar method for producing all the maximal solutions of our inequation. As an example, consider the insertion operation. Recall that the right inverse of insertion is the operation of reversed dipolar deletion, and the left inverse of insertion is deletion. Moreover, [7,8] for every regular language F there exist finitely many languages that can be obtained from F by dipolar deletion, and finitely many languages that can be obtained from F by deletion. This implies that the sets of left and right quotients of F with respect to insertion are finite. Hence, the above method can be applied to solve the inequation $X \longleftarrow X \subseteq R$ with $X \subseteq M$ when R is regular. For example, consider the inequation

$$X \rightleftharpoons' \{aa\} \subseteq X^c.$$

Using the facts $\rightleftharpoons' = \longleftarrow^r$ and $\rightleftharpoons'^r = \longleftarrow$, we can verify that the set $\{\{aa\} \rightleftharpoons W \mid W \subseteq \Sigma^*\}$ consists of all the right quotients of $\rightleftharpoons'^r$ and is equal to the set of all subsets of $\{\lambda, a, aa\}$. Moreover, for each such quotient P_r, say, we can compute the set $\Sigma^* - P_r - (\{aa\} \longrightarrow P_r^c)$, which is equal to $\Sigma^* - P_r - (\{aa\} \rightleftharpoons'^l P_r^c)$ using the fact $\rightleftharpoons'^l = \rightleftharpoons^{r'}$. This process produces two maximal sets, $\{a, aa\}^c$ and $\{\lambda, a\}^c$, which are the maximal solutions of the above inequation—see Corollary 6.12.

7.2. Finite operations

A binary operation is *finite* if its characteristic relation is finite. Finite operations can be obtained by restricting the domain of other operations that are infinite, in general.

Example 7.1. For any positive integer n, let $(\longrightarrow_{rq})_n$ be the restriction of $\longrightarrow_{rq}$ as follows: (w, u, v) is in the characteristic relation of $(\longrightarrow_{rq})_n$ if and only if it is in the characteristic relation of $\longrightarrow_{rq}$ and $|u| \leqslant n$ and $|v| > 0$. Then $\mathrm{dom}_2(\longrightarrow_{rq})_n$ is equal to $\Sigma \cup \cdots \cup \Sigma^n$. Moreover the solution set of the inequation $X(\longrightarrow_{rq})_n \Sigma^+ \subseteq X^c$ with $X \subseteq \Sigma \cup \cdots \cup \Sigma^n$ is the set of all prefix codes whose longest word is of length at most n.

Example 7.2. For any positive integers n and m with $n > m$, let $\bowtie_{n,m}$ be the restriction of $\bowtie$ as follows: (w, u, v) is in the characteristic relation of $\bowtie_{n,m}$ if and only if it is in the characteristic relation of $\bowtie$ and $|u| = n$ and $m \geq |v| > 0$. Then $\mathrm{dom}_2(\bowtie_{n,m})$ is equal to $\Sigma \cup \cdots \cup \Sigma^m$. Moreover the solution set of the inequation $X \bowtie_{n,m} \Sigma^+ \subseteq X^c$ with $X \subseteq \Sigma^n$ is the set of all subsets of Σ^n that are error-detecting for the channel $\sigma_s(m, \infty)$.

In the above examples, the inequation $X\diamond L \subseteq X^c$ with $X \subseteq M$ is such that $\mathrm{dom}_2(\diamond) \subseteq L$. By Corollary 6.5, such an inequation is equivalent to the equation

$$X\diamond^r X = \emptyset \text{ with } X \subseteq M. \tag{$***$}$$

When the operation $\diamond$ and the set M are finite there is an algorithm to test whether $(***)$, hence also $(*)$, has a solution of cardinality k for some given $k \geq 1$—the operation $\diamond$ is given as input by simply listing the elements of $C_\diamond$. The problem, however, is NP-complete.

Proposition 7.3. *The following problem is NP-complete.*
Input: *a finite operation $\diamond$, a finite language M and a positive integer k.*
Output: *Y/N, depending on whether the equation $X\diamond X = \emptyset$ with $X \subseteq M$ has a solution of cardinality k.*

Proof. Firstly, we note that the problem is in NP. Now suppose Σ is the alphabet of the problem. We shall reduce to this problem the following NP-complete problem.
Input: a graph G and a positive integer k.
Output: Y/N, depending on whether G has a clique of k vertices.

Let $G = (V_G, E_G)$ and k constitute an instance of the clique problem, where V_G is the set of vertices and E_G is the set of edges. Suppose $V_G = \{\bar{1}, \ldots, \bar{m}\}$, for some $m \geqslant 1$, where $\bar{v}$ denotes the $|\Sigma|$-ary representation of the integer v using symbols from Σ. Define the finite operation $\diamond_G$ whose characteristic relation consists of all triples $(\lambda, \bar{u}, \bar{v})$ with $\bar{u}, \bar{v} \in V_G$ and $\bar{u} \neq \bar{v}$ and $(\bar{u}, \bar{v})$ is not an edge in E_G. Then the graph G has a clique C of k vertices if and only if C is a solution of the equation $X\diamond_G X = \emptyset$ with $X \subseteq V_G$. This follows by the definition of $\diamond_G$ and the fact that, for every binary operation $\diamond$ and language S, $S\diamond S = \emptyset$ if and only if $(s, t) \notin \mathrm{dom}\,(\diamond)$ for all s and t in S. □

7.3. Decidability of maximality

We discuss now the problem of deciding whether a code of a certain type is maximal using the ideas developed in Proposition 6.6.

The residue of a prefix code S (see Example 3.1) is $\Sigma^+ - (S \cup S \longrightarrow_{rq} \Sigma^+ \cup S\Sigma^+)$ and can be computed in time $O(|A|)$, when S is given by a trim DFA A. This can be done as follows. First, construct a DFA B of size $O(|A|)$ such that $L(B) = S\Sigma^+$. This is possible by adding in A a new state g, which would be the only final state of B, and transitions $fa \to g$ for every $a \in \Sigma$ and for every (old) final state f of g. As $L(A)$ is a prefix code, the automaton B would be a DFA. Now let C be the DFA, of size $O(|A|)$, obtained from B by making all states of B final. Then, it follows that $L(C) = S \cup (S \longrightarrow_{rq} \Sigma^+) \cup S\Sigma^+$. As C is a DFA, we can construct the automaton $A_{\Sigma^+} \cap C^c$ in time $O(|A|)$, where A_{Σ^+} is the two-state automaton accepting Σ^+. The claim follows now, as $L(A_{\Sigma^+} \cap C^c)$ is the residue of S. Hence, we have shown the following consequence of Proposition 6.6.

Corollary 7.4. *The following problem is decidable in linear time*:
Input: *a DFA A.*
Output: *Y/N depending on whether the language $L(A)$ is a maximal prefix code.*

We now turn to the problem of whether a given *finite* suffix (respectively, bifix, infix) code S is a maximal suffix (respectively, bifix, infix) code. As in [3], we assume that the code S is given by listing the words comprising S and, therefore, the size of S, which we denote by $\|S\|$, is equal to $\sum_{u\in S}|u|$. In our discussion, the *trie* T_S of the finite language S plays an important role. This is the trim DFA

$$(\{[p]\mid p\in \mathrm{Pref}\,(S)\},\ \Sigma,\ [\lambda],\ \{[s]\mid s\in S\},\ P)$$

accepting S [3], where $P=\{[p]a\to[pa]\mid p\in\mathrm{Pref}\,(S),\ a\in\Sigma,\ pa\in\mathrm{Pref}\,(S)\}$ and $\mathrm{Pref}\,(S)$ is the set of all prefixes of S. Note that each state $[p]$ represents the prefix p of the input word that has been read so far by the automaton. We shall use the following facts about tries [3] (the alphabet Σ is considered fixed in our paper):

- Given S, the trie T_S is of size $O(\|S\|)$ and can be constructed in time $O(\|S\|)$.
- Given S, one can use the trie T_S to construct a trim DFA D_S, of size $O(\|S\|)$, in time $O(\|S\|)$ accepting the language Σ^*S. The DFA D_S is called the dictionary-matching automaton of S.

First suppose S is a finite suffix code. Then the set S' consisting of the reverses of the words in S is a prefix code. Moreover, S is a maximal suffix code if and only if S' is a maximal prefix code. Hence, to test whether S is a maximal suffix code, one constructs the trie $T_{S'}$ and tests whether $L(T_{S'})$ is a maximal prefix code using Corollary 7.4. Now suppose that S is a *bifix* code—this is a code that is both prefix and suffix. By [1], S is a maximal bifix code if and only if it is a maximal prefix code and a maximal suffix code. Hence, the following holds.

Corollary 7.5. *The following problem is decidable in linear time*:
Input: *a finite language S.*
Output: *Y/N depending on whether S is a maximal suffix, or bifix, code.*

Consider now the case where S is a finite infix code. The residue of S is

$$\begin{aligned}&(S\cup S\rightleftharpoons\Sigma^+\cup S\longleftarrow'\Sigma^+)^c\\ &\quad=(S\cup S\rightleftharpoons\Sigma^+\cup S\cup\Sigma^+\longleftarrow S)^c\\ &\quad=(S\rightleftharpoons\Sigma^*\cup\Sigma^*\longleftarrow S)^c\\ &\quad=(\mathrm{Fact}\,(S)\cup\Sigma^*S\Sigma^*)^c,\end{aligned}$$

where $\mathrm{Fact}\,(S)$ is the set of all factors of S. Given S, one can construct the factor automaton F_S of S that accepts the language $\mathrm{Fact}\,(S)$. This automaton is a minimal DFA of size $O(\|S\|)$ and can be constructed in time $O(\|S\|)$ [3]. We also need to construct a DFA E_S accepting the language $\Sigma^*S\Sigma^*$. For this, consider the dictionary-matching automaton D_S. This has the same states as the trie T_S does, the same final states, and includes all the productions of T_S. In addition, for each state $[p]$ of D_S and for each symbol $a\in\Sigma$, if there is no production of the form $[p]a\to[pa]$ in T_S then we add in D_S the production $[p]a\to[u]$ where u is the longest suffix of pa that is also a prefix of S (hence, $[u]$ would be a valid state of T_S and D_S). To obtain the desired DFA E_S we modify slightly the construction of D_S from T_S as follows:

- Add the new productions $[p]a\to[u]$ as specified above, unless $[p]$ is a final state.
- Add an extra final state G in E_S and the productions $Ga\to G$, for all $a\in\Sigma$. Moreover, for every (existing) final state $[w]$ and for every symbol $a\in\Sigma$, add the production $[w]a\to G$.

We argue now that $L(E_S)=\Sigma^*S\Sigma^*$. First consider any word z in $\Sigma^*S\Sigma^*$. This word can be written as xy with $x\in\Sigma^*S$. Thus, there is a successful computation of the automaton D_S on x which involves the sequence of states $[u_0], [u_1], \ldots, [u_n]$, say, where $u_0 = \lambda$. Let $[u_k]$ be the first occurrence of a final state of D_S in the above sequence of states, and let ξ be the computation of D_S corresponding to the sequence $[u_0], \ldots, [u_k]$. In this computation the automaton reads a prefix x_1 of x and, therefore x is of the form x_1x_2. By the construction of E_S, ξ must be a computation of E_S as well and, as $[u_k]$ is a final state, the word x_2y will be accepted by E_S when $[u_k]$ is used as the start state. Hence, x_1x_2y would be accepted by E_S when $[\lambda]$ is used as the start state.

Now consider a word z in $L(E_S)$. There is a computation of E_S that involves a sequence of states $q_0, q_1, \ldots, q_n$ with $q_0 = [\lambda]$. Moreover, there is a unique state q_i that is a final state of D_S such that all states $q_0, \ldots, q_i$ are different from G and, if $i < n$, all states $q_{i+1}, \ldots, q_n$ are equal to G. Then in the computation ξ, say, that corresponds to the states $q_0, \ldots, q_i$ the automaton reads a prefix x_1 of z. But ξ is also a computation of D_S which implies that $x_1 \in \Sigma^* S$ and, therefore, $z \in \Sigma^* S \Sigma^*$ as required.

We return now to the original question of computing the residue of S. According to the above, the residue of S is the language accepted by the automaton $(F_S \sqcup E_S)^c$, which is of size $O(\|S\|^2)$. Hence, we have shown the following.

Corollary 7.6. *The following problem is decidable in quadratic time*:
Input: *a finite language S.*
Output: *Y/N, depending on whether S is a maximal infix code.*

We conclude the paper with the following consequence of Corollary 6.7.

Corollary 7.7. *The following problem is decidable in time $O(\|C\| \log \|C\|)$.*
Input: *Fixed-length code C that is error-detecting for the channel $\sigma_s(1, \infty)$.*
Output: *Y/N, depending on whether C is a maximal subset of Σ^n with the property of being error-detecting for the channel $\sigma_s(1, \infty)$, where n is the length of the words in C.*

Proof. By Example 4.4, the above problem is equivalent to deciding whether the solution C of

$$X \bowtie \Sigma \subseteq X^c \quad \text{with} \quad X \subseteq \Sigma^n$$

is maximal, and by Lemma 4.6 and Remark 4.7, the residue of the solution C is equal to $\Sigma^n - C \bowtie (\Sigma^0 \cup \Sigma)$. Moreover, as $C \bowtie (\Sigma^0 \cup \Sigma) \subseteq \Sigma^n$ it follows that C is maximal if and only if the cardinality of $C \bowtie (\Sigma^0 \cup \Sigma)$ is q^n, where $q = |\Sigma|$.

Now note that, if C is maximal, then it must be the case that $|C| + |C|(q-1)n \geqslant q^n$. This follows from the fact that $C \bowtie (\Sigma^0 \cup \Sigma)$ is equal to $C \cup (\bigcup_{w \in C} w \bowtie \Sigma)$ and the cardinality of each $w \bowtie \Sigma$ is $(q-1)n$. This implies that, if $|C|(1 + (q-1)n) < q^n$, then C is not maximal. Based on these observations, we have the following decision procedure.

(i) Let n be the length of the words in C and let q be the cardinality of the alphabet.
(ii) If $|C|(1 + (q-1)n) < q^n$ then output **N** and quit.
(iii) Initialize a set of words S to C and a counter to $|C|$.
(iv) For each word w in C, compute the $(q-1)n$ words of $w \bowtie \Sigma$ and insert them in S. Moreover, increment the counter by one each time a *new* word is inserted in S. If the counter becomes q^n output **Y** and quit.
(v) Output **N**.

Obviously, the worst case time complexity of the algorithm is dominated by steps 3 and 4. We can implement S as a trie T, which is initialized to T_C. The cost of inserting a word of length n in a trie is $\Theta(n)$. Hence, the cost of steps 3 and 4 is

$$\Theta(\|C\|) + \Theta(|C| \times (q-1)n \times n),$$

which is equivalent to $\Theta(\|C\|(q-1)n)$ using the fact that $\|C\| = n|C|$. Also, in these steps we have that $q^n \leqslant |C|(1+(q-1)n)$, which implies

$$q^n \leqslant |C|qn \Rightarrow q^{n-1} \leqslant \|C\| \Rightarrow n-1 \leqslant \log_q \|C\|$$

and the claim of the corollary is established. □

Acknowledgements

The authors are thankful to the referees whose comments helped in improving the presentation of the results.

References

[1] J. Berstel, D. Perrin, Theory of Codes, Academic Press, Orlando, 1985.
[2] J.H. Conway, Regular Algebra and Finite Machines, Chapman & Hall, London, 1971.
[3] M. Crochemore, C. Hancart, Automata for Matching Patterns, in: G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, vol. II, Springer, Berlin, 1997, pp. 399–462.
[4] T. Head, A. Weber, Deciding code related properties by means of finite transducers, in: R. Capocelli, A. de Santis, U. Vaccaro (Eds.), Sequences II, Methods in Communication, Security, and Computer Science, Springer, Berlin, 1993, pp. 260–272.
[5] H. Jürgensen, S. Konstantinidis, Codes, in: G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, vol. I, Springer, Berlin, 1997, pp. 511–607.
[6] H. Jürgensen, K. Salomaa, S. Yu, Transducers and the decidability of independence in free monoids, Theoret. Comput. Sci. 134 (1994) 107–117.
[7] L. Kari, On insertion and deletion in formal languages, Ph.D. Thesis, University of Turku, Finland, 1991.
[8] L. Kari, On language equations with invertible operations, Theoret. Comput. Sci. 132 (1994) 129–150.
[9] L. Kari, G. Thierrin, Maximal and minimal solutions to language equations, J. Comput. System Sci. 53 (1996) 487–496.
[10] S. Konstantinidis, Relationships between different error-correcting capabilities of a code, IEEE Trans. Inform. Theory 47 (2001) 2065–2069.
[11] S. Konstantinidis, Transducers and the properties of error-detection, and finite-delay decodability, J. Universal Comp. Sci. 8 (2002) 278–291.
[12] S. Konstantinidis, A. O'Hearn, Error-detecting properties of languages, Theoret. Comput. Sci. 276 (2002) 355–375.
[13] G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, Springer, Berlin, 1997.
[14] A. Salomaa, Formal Languages, Academic Press, London, 1973.
[15] H.J. Shyr, Free Monoids and Languages, Institute of Applied Mathematics, National Chung-Hsing University, Taichung, Taiwan, 1991.
[16] S. Yu, Regular Languages, in: G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, vol. I, Springer, Berlin, 1997, pp. 41–110.