Applications of Transducers in Independent Languages, Word Distances, Codes

Stavros Konstantinidis

Applications of Transducers in Independent Languages, Word Distances, Codes*

Stavros Konstantinidis

Department of Mathematics and Computing Science,
Saint Mary's University, 923 Robie Str, Halifax, Nova Scotia, B3H 3C3, Canada
s.konstantinidis@smu.ca

Abstract. A (nondeterministic) transducer t is an operator mapping an input word to a set of possible output words. A few types of transducers are important in this work: input-altering, input-preserving, and input-decreasing. Two words are t-dependent, if one is the output of t when the other one is used as input. A t-independent language is one containing no two t-dependent words. Examples of independent languages are found in noiseless coding theory, noisy coding theory and DNA computing. We discuss how the above transducer types can provide elegant solutions to some cases of the following broad problems: (i) computing two minimum distance witness words of a given regular language; (ii) computing witness words for the non-satisfaction, or non-maximality, of a given regular language with respect to the independence specified by a given transducer t; (iii) computing, for any given t and language L, a maximal t-independent language containing L; (iv) computing, for any given positive integer n and transducer t, a t-independent language of n words. The descriptional complexity cost of converting between transducer types is discussed, when this conversion is possible. We also explore methods of defining more independences in a way that some of the above problems can still be computed.

Keywords: algorithm, automata, codes, distance, independence, language, maximal, transducer

1 Introduction

The Abstract already serves as the first part of the introduction to this paper. Our main objective is to survey results related to describing independences and answering algorithmic questions about such independences. In particular, the description of an independence can be part of the input to the algorithms of interest. Many independences are known as *code-related properties*. The paper is organized as follows. Sect. 2 contains the basic terminology. Sect. 3 explains the use of transducers to describe the kind of independences we are interested in. Sect. 4 presents the algorithmic questions we consider and discusses how to

* Research supported by NSERC.

solve some of those questions. Sect. 5 presents results about embedding a given independent language to a maximal one when the independence is described by an input-decreasing transducer. Sect. 6 discusses a method of computing minimum distance witnesses of a given regular language. Sect. 7 discusses some known facts about converting one type of transducer to another. Sect. 8 explores new ways of describing independences. Sect. 9 contains a few concluding remarks.

At the end of some sections we also discuss directions for further research.

2 Terminology

We write $\mathbb{N}, \mathbb{N}_0$ for the sets of positive integers and non-negative integers, respectively. If S is a set, then $|S|$ denotes the cardinality of S, and 2^S denotes the set of all subsets of S. When there is no risk of confusion, we write a singleton set $\{x\}$ simply as x. Thus, $S \cup x$ means $S \cup \{x\}$. An ***alphabet*** is a finite nonempty set of symbols. We write Σ, Δ for arbitrary alphabets. The set of all words, or strings, over Σ is denoted by Σ^*, which includes the ***empty*** word λ. As usual, Σ^+ denotes $\Sigma^* - \lambda$. A ***language*** (over Σ) is any set of words. Let L be a language and let u, v, w, x be any words. If $w \in L$ then we say that w is an ***L-word***. We use standard operations and notation on words and languages, [14,42,30], in particular $|w|$ denotes the length of w and $\overline{L}$ denotes $\Sigma^* - L$. If w is of the form uv then u is a ***prefix*** and v is a ***suffix*** of w. If $u \neq w$ then u is called a ***proper*** prefix of w—the definition of proper suffix is analogous. A ***relation*** over Σ and Δ is a subset of $\Sigma^* \times \Delta^*$, that is, a set of pairs (x, y) of words over the two alphabets (respectively). The ***inverse*** of ρ, denoted by ρ^{-1}, is the relation $\{(y, x) \mid (x, y) \in \rho\}$.

Independent languages, code properties. Consider a fixed, but arbitrary, alphabet Σ containing at least two symbols. A (language) property is any set of languages, that is, subsets of Σ^*. An ***independence***, or ***code property***, [17], is a property $\mathcal{P}$ such that $L \in \mathcal{P}$, if and only if, $L' \in \mathcal{P}$ for all $L' \subseteq L$ with $0 < |L'| < n$, for some $n \in \mathbb{N} \cup \{\aleph_0\}$. If $L \in \mathcal{P}$ then we say that L ***satisfies*** $\mathcal{P}$. Thus, L satisfies $\mathcal{P}$ exactly when all nonempty subsets of L with less than n elements satisfy $\mathcal{P}$. In this case, we also say that $\mathcal{P}$ is an ***n-independence***. A language $L \in \mathcal{P}$ is called ***$\mathcal{P}$-maximal***, or a maximal $\mathcal{P}$ code, if $L \cup w \notin \mathcal{P}$ for any word $w \notin L$. Every language satisfying $\mathcal{P}$ is included in some $\mathcal{P}$-maximal language [17]. To our knowledge, almost all code related properties in the literature [12,38,17,44,4,15,8,22,31,18,9,34,45,20] are independences. Also, any independence with respect to a binary relation in the sense of [39] is a 3-independence. Examples: (i) L is a ***prefix code*** if no L-word is a prefix of another L-word—this is a 3-independence; (ii) L is ***2-substitution error-detecting*** if no L-word can result by changing one or two symbols in some other L-word—this again is a 3-independence; (iii) L is a ***UD-code***[1], if every word in L^+ can be parsed in exactly one way as a list of words in L—this is an $\aleph_0$-independence.

[1] UD means Uniquely Decodable/Decipherable.

Automata and regular languages [43,37]. A nondeterministic finite automaton (*NFA*) is a quintuple $\mathbf{a} = (Q, \Sigma, T, I, F)$ such that Q is the set of states, Σ is an alphabet, $I, F \subseteq Q$ are the sets of start (or initial) states and final states, respectively, and $T \subseteq Q \times \Sigma \times Q$ is the finite set of *transitions*. If (p, x, q) is a transition, then x is the *label* of the transition, and we say that p has an *outgoing* transition (with label x). The *language accepted* by $\mathbf{a}$, denoted by $\mathrm{L}(\mathbf{a})$, is the set of words w formed by concatenating the labels that occur in some accepting path of $\mathbf{a}$. The *size* $|\mathbf{a}|$ of the automaton $\mathbf{a}$ is $|Q| + |T|$.

Transducers [3,43,37]. A *transducer* is a sextuple $\mathbf{t} = (Q, \Sigma, \Delta, T, I, F)$ such that Q, I, F are exactly the same as those in NFAs, Σ is now called the *input* alphabet, Δ is the *output* alphabet, and $T \subseteq Q \times \Sigma^* \times \Delta^* \times Q$ is the finite set of transitions. We write $(p, x/y, q)$ for a transition—its *label* (x/y) consists of the input label x and the output label y. The relation $\mathrm{R}(\mathbf{t})$ *realized* by the transducer $\mathbf{t}$ is the set of word pairs (u, v) such that u (resp. v) is formed by concatenating the input labels (resp. the output labels) that occur in some accepting path of $\mathbf{t}$. We write $\mathbf{t}(u)$ for the set of *possible outputs* of $\mathbf{t}$ on input u, that is, $v \in \mathbf{t}(u)$ iff $(u, v) \in \mathrm{R}(\mathbf{t})$. For any language L, $\mathbf{t}(L)$ is the language $\cup_{u \in L} \mathbf{t}(u)$. The *domain* of $\mathbf{t}$ is the set of all words u such that $\mathbf{t}(u) \neq \emptyset$. The transducer $\mathbf{t}$ is called *functional* if, for all words u, $\mathbf{t}(u)$ contains at most one word. Examples of transducers are shown in Fig. 1. The *size* of a transition $(p, x/y, q)$ is the number $1 + |x| + |y|$. The size $|\mathbf{t}|$ of the transducer $\mathbf{t}$ is the number of states plus the sum of the sizes of the transitions in T. For any transducers $\mathbf{s}$, $\mathbf{t}$ and NFA $\mathbf{a}$, we have the following. There is a transducer $\mathbf{t}^{-1}$ of size $O(|\mathbf{t}|)$ realizing the relation $\big(\mathrm{R}(\mathbf{t})\big)^{-1}$. There is a transducer $\mathbf{s} \vee \mathbf{t}$ of size $O(|\mathbf{s}| + |\mathbf{t}|)$ realizing $\mathrm{R}(\mathbf{s}) \cup \mathrm{R}(\mathbf{t})$. There are transducers $\mathbf{t} \downarrow \mathbf{a}$ and $\mathbf{t} \uparrow \mathbf{a}$, each of size $O(|\mathbf{t}| \cdot |\mathbf{a}|)$, realizing the relations $\{(u, v) : u \in \mathrm{L}(\mathbf{a}), v \in \mathbf{t}(u)\}$ and $\{(u, v) : u \in \Sigma^*, v \in \mathrm{L}(\mathbf{a}) \cap \mathbf{t}(u)\}$, respectively [23].

3 t-Independent Languages: classic & antimorphic

The concept of ρ-independence, where ρ is a binary relation, goes back to [39]—see also [38,44]. Here we rephrase this concept in terms of transducers, with the aim of answering *algorithmic questions* about independences using transducer tools—see Sect. 4. We focus on certain transducer types that turn out to be important in the context of those algorithmic questions. First we consider "classic" independences, and then (see further below) "antimorphic" ones.

Definition 1. *Let '$\prec$' be the lexicographic word order*[2]*. A transducer* $\mathbf{t}$ *is*

input-preserving, if $x \in \mathbf{t}(x)$*, for all words* x *in the domain of* $\mathbf{t}$*;*
input-altering, if $x \notin \mathbf{t}(x)$*, for all words* x*;*
length-decreasing, if $y \in \mathbf{t}(x)$ *implies* $|y| < |x|$*, for all words* x, y*;*
input-decreasing, if $y \in \mathbf{t}(x)$ *implies* $y \prec x$*, for all words* x, y*.*

[2] In this order, the alphabet Σ is totally ordered. Then, $u \prec v$ if and only if, either $|u| < |v|$, or $|u| = |v|$ and $u = x\sigma_1 y_1, v = x\sigma_2 y_2$ such that $\sigma_1, \sigma_2 \in \Sigma$ and $\sigma_1 \prec \sigma_2$.

Remark 1. Every length-decreasing transducer is input-decreasing and every input-decreasing transducer is input-altering. Moreover, for every transducer $\mathbf{t}$, the transducer $(\mathbf{t} \vee \mathtt{eq})$ is input-preserving—see Fig. 1.

Definition 2. *Let* $\mathbf{t}$ *be a transducer. A language* L *is called* $\mathbf{t}$*-independent, if the following condition holds for all words* u, v*:*

$$u, v \in L \text{ and } v \in \mathbf{t}(u) \text{ implies } u = v. \tag{1}$$

$\mathcal{P}_{\mathbf{t}}$ *denotes the independence described by* $\mathbf{t}$*; this is the set of all languages satisfying the above condition. In [9], if* $\mathbf{t}$ *is input-preserving then* $\mathcal{P}_{\mathbf{t}}$ *is called an input-preserving transducer property, and if* $\mathbf{t}$ *is input-altering then* $\mathcal{P}_{\mathbf{t}}$ *is called an input-altering transducer property. If the transducer* $\mathbf{t}$ *is input-altering, then condition* (1) *is equivalent to*

$$\mathbf{t}(L) \cap L = \emptyset \tag{2}$$

Remark 2. One verifies that condition (1) is equivalent to the one resulting if we replace $\mathbf{t}$ with $\mathbf{t}^{-1}$, or with $(\mathbf{t} \vee \mathbf{t}^{-1})$. Similarly, one verifies that condition (2) is equivalent to "$\mathbf{t}^{-1}(L) \cap L = \emptyset$", and also to "$(\mathbf{t} \vee \mathbf{t}^{-1})(L) \cap L = \emptyset$".

Remark 3. Every independence described by a transducer is a 3-independence. Moreover, for every transducer $\mathbf{t}$, every singleton language $\{w\}$ is $\mathbf{t}$-independent.

Remark 4. The type of transducer used to describe an independence affects the computability and complexity of algorithmic questions about that independence (see subsequent sections). It turns out, however, that limiting (1) to $\mathbf{t}$'s that are input-preserving does not constitute any restriction on the describable independences. Indeed, one can confirm that, for any transducer $\mathbf{t}$, we have that

$$\text{the transducer } (\mathbf{t} \vee \mathtt{eq}) \text{ is input-preserving and } \mathcal{P}_{\mathbf{t}} = \mathcal{P}_{\mathbf{t} \vee \mathtt{eq}}.$$

Remark 5. In [25], the independence described by an input-preserving transducer $\mathbf{t}$ is called an *error-detecting* property. Specifically, $\mathbf{t}$ can be viewed as a communication channel such that, if $v \in \mathbf{t}(u)$ and $v \neq u$ then we have a channel error. A language satisfying $\mathcal{P}_{\mathbf{t}}$ is called error-detecting *for* $\mathbf{t}$. This means that the channel $\mathbf{t}$ cannot turn an L-word to a different L-word.

Example 1. The transducers $\mathtt{px}, \mathtt{sx}, \mathtt{hc}$ shown in Fig. 1 are input-altering and describe, respectively, prefix codes, suffix codes, and hypercodes. A language L is a *hypercode* if no L-word results by deleting at least one symbol in some other L-word, that is, $\mathtt{hc}(L) \cap L = \emptyset$. The transducer $\mathtt{sub}_1$ is input-preserving and describes the 1-substitution error-detecting languages. The transducer $\mathtt{id}_2$ is input-preserving and describes the 2-synchronization error-detecting languages. A language L is k*-synchronization* error-detecting if no L-word results by inserting and/or deleting a total of at most k symbols in some other L-word. For example, $\{0101, 1010\}$ is not 2-synchronization error-detecting as 1010 results by deleting the first 0 of $u = 0101$ and inserting a 0 at the end of u. Using $\mathtt{sub}_1, \mathtt{id}_2$, as a guide, one can design input-preserving transducers $\mathtt{sub}_k, \mathtt{id}_k$ describing the k-substitution and k-synchronization error-detecting languages, for any $k \in \mathbb{N}$.

Remark 6. In [8], 3-independences are defined by trajectories. A regular trajectory expression $\bar{e}$ is any regular expression over $\{0,1\}$ and describes the independence consisting of all languages L such that $(L \sqcup\!\sqcup_{\bar{e}} \Sigma^+) \cap L = \emptyset$, where $\sqcup\!\sqcup_{\bar{e}}$ is the shuffle operation according to $\bar{e}$. For example, 0^*1^* describes prefix codes because the shuffle inserts symbols at the end of an L-word and the result cannot be an L-word. In [9], it is shown that, for every regular trajectory expression, there is an input-altering transducer describing the same independence. Moreover, there are natural error-detecting properties that cannot be described by regular trajectory expressions. On the other hand, the method of trajectories provides a simple and effective method for describing many independences.

Antimorphic Independences. Independences related to DNA computing cannot be described by transducers as in Definition 2, [20]. This is because such independences involve the antimorphic involution "dna" over the DNA alphabet $\{\mathtt{a},\mathtt{c},\mathtt{g},\mathtt{t}\}$. In general, let θ be a permutation on the alphabet Σ. Then, θ is called an ***involution*** if $\theta^{-1} = \theta$. A ***morphic*** permutation on Σ^* is a permutation on Σ extended to Σ^* such that $\theta(\lambda) = \lambda$ and $\theta(\sigma x) = \theta(\sigma)\theta(x)$, for all $\sigma \in \Sigma$ and $x \in \Sigma^*$. When θ is extended such that $\theta(\sigma x) = \theta(x)\theta(\sigma)$, then it is called an ***antimorphic*** permutation. For example, the antimorphic involution "dna" is such that $\mathrm{dna}(\mathtt{a}) = \mathtt{t}$ and $\mathrm{dna}(\mathtt{c}) = \mathtt{g}$. Let θ be any (anti)morphic permutation and let $\mathbf{t}$ be any transducer. In [20], the independence ***described by*** $\mathbf{t}$ ***and*** θ is the set of all languages L such that

$$\mathbf{t}(L) \cap \theta(L) = \emptyset. \tag{3}$$

Example 2. A language L is called ***strictly*** θ***-compliant***, [19,20], if it contains no two words of the form $x\theta(v)y$ and v. This is equivalent to saying that by deleting a prefix and/or a suffix of an L-word (see transducer $\mathtt{sco}$) the resulting word cannot be in $\theta(L)$, that is, $\mathtt{sco}(L) \cap \theta(L) = \emptyset$; hence, strict θ-compliance is described by $\mathtt{sco}$ and θ. Now consider the independence $\mathcal{H} = \{L \subseteq \Sigma^* : \delta_{\mathrm{H}}(u, \theta(v)) \geq 2, \text{ for all } u, v \in L\}$, [20], where $\delta_{\mathrm{H}}(x, y)$ is the Hamming distance between the words x, y. Then, $\mathcal{H}$ is described by $\mathtt{sub}_1$ and θ.

4 Algorithmic Questions about Independent Languages

In the context of the research on languages and independences, we consider the following algorithmic questions.

Satisfaction question. Given the description of an independence $\mathcal{P}$ and the description of a language, decide whether the language satisfies $\mathcal{P}$. In the ***witness*** version of this problem, a negative answer is also accompanied by an appropriate list of words showing how the property $\mathcal{P}$ is violated.

Maximality question. Given the description of an independence $\mathcal{P}$ and the description of a language L, decide whether the language is $\mathcal{P}$-maximal. In fact we allow the more general problem, where the input includes also the

description of a second language M and the question is whether there is no word $w \in M - L$ such that $L \cup w$ satisfies $\mathcal{P}$. The default case is when $M = \Sigma^*$. In error control coding (see also the construction question below) $M = \Sigma^\ell$, for some positive integer ℓ, so all languages of interest are ***block codes***, that is, their words are of fixed length ℓ. In the ***witness*** version of this question, a negative answer is also accompanied by any word $w \in M - L$ that can be added to the language L.

Embedding question. Given the description of an independence $\mathcal{P}$ and the description of a language L satisfying $\mathcal{P}$, compute a $\mathcal{P}$-maximal language containing L. As above, maximality can be considered with respect to a certain language M.

Construction question. Given the description of an independence $\mathcal{P}$ and two positive integers n and ℓ, construct a language that satisfies $\mathcal{P}$ and contains n words of length ℓ (if possible).

Minimum distance question. Given the description of a language L, compute the minimum distance between any two different L-words. In the ***witness*** version, also return two minimum distance L-words. This question depends on the distance of interest—see Sect. 6.

In this section, we assume that an independence is described by a transducer $\mathbf{t}$, and possibly by a permutation θ, and in the first three problems, the language is given via an NFA $\mathbf{a}$. In the maximality question, the second language M is also given via an NFA $\mathbf{b}$. Answers to the embedding and minimum distance questions are discussed in the next sections.

Remark 7. Next we list a few algorithmic tools used in answering some of the above questions. Occasionally we use *object-oriented notation:* $\mathbf{t}.f(P)$ is the algorithm f of the object $\mathbf{t}$ with list of parameters P.

1. $\mathbf{t}$.`nEmptyW`(), where $\mathbf{t}$ is a transducer, returns either a pair of words in $\mathrm{R}(\mathbf{t})$, or (`None`, `None`) if $\mathrm{R}(\mathbf{t}) = \emptyset$. Similarly, for any NFA $\mathbf{a}$, we have that $\mathbf{a}$.`nEmptyW`() returns either a word in $\mathrm{L}(\mathbf{a})$ or `None`. This algorithm is called non-emptiness witness, and is based on the standard shortest path algorithm from the start states to the final states of $\mathbf{t}$ or $\mathbf{a}$.
2. $\mathbf{t}$.`nFunctW`() returns either a triple (u, v_1, v_2) of words such that $v_1 \neq v_2$ and $v_1, v_2 \in \mathbf{t}(u)$, or a triple of `None`'s if the transducer $\mathbf{t}$ is functional. This algorithm is a witness version, [25], of the decision algorithm for transducer functionality in [2,1].
3. $(\mathbf{a} \triangleright \mathbf{t})$ is an NFA accepting the language $\mathbf{t}\big(\mathrm{L}(\mathbf{a})\big)$, for any given transducer $\mathbf{t}$ and NFA $\mathbf{a}$.

Some results from [9,20,26] are the following.

1. The witness version of the satisfaction question is polynomially computable and, in fact, reasonably efficiently so—see the above references. More specifically we have the following cases: (i) given transducer $\mathbf{t}$ (and possibly a permutation θ) and NFA $\mathbf{a}$, return either (`None`, `None`) if $\mathrm{L}(\mathbf{a})$ satisfies (3), or a pair (u, v) of $\mathrm{L}(\mathbf{a})$-words such that (3) is not satisfied for $L = \{u, v\}$;

(ii) given input-preserving transducer $\mathbf{t}$ and NFA $\mathbf{a}$, return as above, either $(\texttt{None}, \texttt{None})$, or a pair (u, v) depending on whether (1) is satisfied. Case (i) is solved using the algorithm $(\mathbf{t} \downarrow \mathbf{a} \uparrow \mathbf{a}^{\theta}).\texttt{nEmptyW}()$, where $\mathbf{a}^{\theta}$ is an NFA accepting $\theta(\mathrm{L}(\mathbf{a}))$. For case (ii), condition (1) is equivalent to whether

$$\text{the transducer } \ (\mathbf{t} \downarrow \mathbf{a} \uparrow \mathbf{a}) \ \text{ is functional.} \tag{4}$$

Then, the desired witness version is solved by making use of the algorithm $(\mathbf{t} \downarrow \mathbf{a} \uparrow \mathbf{a}).\texttt{nFunctW}()$.

2. The decision version of the maximality question can be coNP-hard, [26], PSPACE-hard or undecidable, [20]. Specifically, if the given transducer $\mathbf{t}$ is input-preserving, or input-altering, or θ-input-altering[3], then deciding whether $\mathrm{L}(\mathbf{a})$ is maximal satisfying, respectively, (1) or (2) or (3), is PSPACE-hard. In fact, the witness version is computable using the algorithm

$$\Big(\mathbf{b} \wedge \big((\mathbf{a} \rhd \mathbf{t}) \vee (\mathbf{a}^{\theta} \rhd \mathbf{t}^{-1}) \vee \mathbf{a}\big)^{\mathrm{co}}\Big).\texttt{nEmptyW}(), \tag{5}$$

where we assume that, for any given NFAs $\mathbf{c}$ and $\mathbf{d}$, $\mathbf{c}^{\mathrm{co}}$ is an NFA accepting the complement of $\mathrm{L}(\mathbf{c})$, $(\mathbf{c} \wedge \mathbf{d})$ is an NFA accepting $\mathrm{L}(\mathbf{c}) \cap \mathrm{L}(\mathbf{d})$, and $(\mathbf{c} \vee \mathbf{d})$ is an NFA accepting $\mathrm{L}(\mathbf{c}) \cup \mathrm{L}(\mathbf{d})$. On the other hand, even for any ***fixed*** permutation θ, it is undecidable to tell, given $\mathbf{t}$ and $\mathbf{a}$, whether $\mathrm{L}(\mathbf{a})$ is maximal satisfying (3), where there is no restriction on $\mathbf{t}$.

3. The approach of [26] for the construction question is as follows: a definition is given of what an $f\%$ $\mathcal{P}$-maximal block code is, and then a simple randomized algorithm is described that is given an input-preserving transducer $\mathbf{t}$ and positive integers n, ℓ, and returns either a $\mathbf{t}$-independent language of n words of length ℓ, or a $\mathbf{t}$-independent language L of ***less*** than n words of length ℓ such that L is a 95% $\mathcal{P}_{\mathbf{t}}$-maximal block code.

5 The Embedding Question

The embedding question has been addressed well for several fixed code properties like UD codes [10], bifix codes [46], solid codes [29], and bounded deciphering delay codes [5]. In [40], the question is solved for any independence described by some length-decreasing-and-transitive transducer. In [24], the embedding question is addressed for any independence described by some input-decreasing transducer. We consider maximality with respect to a certain ***fixed***, but arbitrary, language $\texttt{M}$. We discuss next some recent results from [24].

Definition 3. *A (language)* ***operator*** *is a function* $\mathrm{Op} : 2^{\Sigma^*} \to 2^{\Sigma^*}$. *It is called* ***union respecting*** *if* $\mathrm{Op}(X) = \cup_{v \in X} \mathrm{Op}(v)$, *for all languages* X. *For any language* X *and nonnegative integer* i, *we define the following operators.*

$$\mathrm{Op}^0(X) = X \ \text{ and } \ \mathrm{Op}^{i+1}(X) = \mathrm{Op}(\mathrm{Op}^i(X))$$

$$\mathrm{Op}^*(X) = \cup_{i=0}^{\infty} \mathrm{Op}^i(X), \quad \mathrm{Op}^{\cap}(X) = \cap_{i=1}^{\infty} \mathrm{Op}^i(X)$$

[3] This is when $\theta(w) \notin \mathbf{t}(w)$, for all nonempty words w.

The operator Op *is called* ***exhaustive****, if* $\mathrm{Op}^{\cap}(X) = \emptyset$*, for all languages* X*. If* Op_1 *is also a language operator then we write* $\mathrm{Op} \subseteq \mathrm{Op}_1$ *to indicate that* $\mathrm{Op}(X) \subseteq \mathrm{Op}_1(X)$ *for all languages* X*. A union respecting operator* Op *is* ***functional****, if* $\mathrm{Op}(v)$ *contains at most one word, for all words* v*.*

Example 3. We view a transducer $\mathbf{t}$ as a language operator, which is union respecting. Then, we have that $\mathbf{t}$ is ***transitive*** if and only if $\mathbf{t}^2 \subseteq \mathbf{t}$.

In the sequel we assume that $\mathbf{t}$ is a ***fixed***, but arbitrary, input-altering transducer. We define the following language operators.

$$\mathrm{I}_{\mathbf{t}}(X) = \mathtt{M} - \big(\mathbf{t}(X) \cup \mathbf{t}^{-1}(X)\big) \quad \text{and} \quad \mu_{\mathbf{t}} X = \mathrm{I}_{\mathbf{t}}(X) - \mathbf{t}^{-1}\big(\mathrm{I}_{\mathbf{t}}(X)\big)$$

The above operators are translated to transducer notation from the corresponding ones in [40]. The operator $\mathrm{I}_{\mathbf{t}}(\cdot)$ is the set of all possible words that are either in X or $\mathbf{t}$-independent from X, so in some sense it is the ***max***imum set in which X can be embedded. However, two words in $\mathrm{I}_{\mathbf{t}}(X) - X$ might be $\mathbf{t}$-dependent. The operator mapping any Y to $Y - \mathbf{t}^{-1}(Y)$ is the '$\mathbf{t}$-***min***imize' operator which returns all Y-elements that cannot produce another Y-element via $\mathbf{t}$.

Definition 4. *The operator* $\mu_{\mathbf{t}}$ *is called the* ***max-min operator****. The operator* $\mu_{\mathbf{t}}^*$ *is called the* ***iterated*** *max-min operator. We say that it* ***converges finitely*** *on a language* L*, if there is* $i \in \mathbb{N}_0$ *such that* $\mu_{\mathbf{t}}^* L = \mu_{\mathbf{t}}^i L$*.*

In the case of codes defined by length-decreasing-and-transitive transducers, already the language $\mu_{\mathbf{t}} L$ is maximal and constitutes a solution to the embedding question, [40], where L is the given language satisfying $\mathcal{P}_{\mathbf{t}}$. As stated in [40], however, this does not work for other codes like bifix codes, and also for error-detecting codes. The following results about any $\mathbf{t}$-independent language L are shown in [24].

1. If $\mathbf{t}^{-1}$ is exhaustive and there is $i \in \mathbb{N}_0$ such that $\mu_{\mathbf{t}}^{i+1} L = \mu_{\mathbf{t}}^i L$ then $\mu_{\mathbf{t}}^i L$ is $\mathcal{P}_{\mathbf{t}}$-maximal containing L.
2. If $\mathbf{t}^{-1}$ is exhaustive and $\mathbf{t}$ is transitive then $\mu_{\mathbf{t}} L$ is $\mathcal{P}_{\mathbf{t}}$-maximal containing L.
3. If $\mathbf{t}$ is input-decreasing then $\mu_{\mathbf{t}}^* L$ is $\mathcal{P}_{\mathbf{t}}$-maximal containing L. Note that if $\mathbf{t}$ is input-decreasing then $\mathbf{t}^{-1}$ is exhaustive.

Example 4. [24] Let $\mathtt{M} = \{0,1\}^*$ and let $\mathbf{t} = (\mathtt{px} \vee \mathtt{sx})$, an input-decreasing transducer describing ***bifix codes***. We have that $\mu_{\mathbf{t}}(001) = \{001, 000, 10, 11\}$ and $\mu_{\mathbf{t}}^2(001) = 01^*0(0+1) + 10 + 11$, which is maximal. Again, we have $\mu_{\mathbf{t}}^2\big((0+1)^3 11\big) = (0+1)^3(0+10^*1)$, which is maximal—this code is the reverse of what [4] calls the reversible Golomb-Rice code. Now let $\mathtt{M} = \{0,1\}^5$ and $\mathbf{t} = \mathtt{sub}_1^{\prec}$. Then, the following code is maximal and known as the even-parity code of length 5: $\mu_{\mathbf{t}}^3(01111) = \{w \in \{0,1\}^5 \mid w\text{'s count of 1s is even}\}$.

Research Directions. Statement 3 above can be applied to the case where $\mathbf{t}$ describes bifix codes, hence, if L is a bifix code then $\mu_{\mathbf{t}}^* L$ is a maximal bifix code containing L. An open question stated in [24] is whether in this case $\mu_{\mathbf{t}}^*$ converges finitely. We believe that the answer here is yes, so this would give an alternate

proof of the fact that every regular bifix code can be embedded in a maximal one [46,4]. Computing $\mu_{\mathbf{t}} L(\mathbf{a})$, for any given transducer $\mathbf{t}$ and NFA $\mathbf{a}$, requires complementing an NFA, which is known to be a hard problem. The question here is whether there is a heuristic that could compute efficiently "chunks" of $\mu_{\mathbf{t}} L(\mathbf{a})$ containing $L(\mathbf{a})$ and somehow these chunks get "close" to being maximal as $\mu_{\mathbf{t}}$ gets iterated.

6 The Minimum Distance Question

An ***integral difference*** is a function δ that maps any pair of words into $\mathbb{N}_0 \cup \{\infty\}$ such that $\delta(x, y) = 0$ if and only if $x = y$. In addition, if $\delta(x, y) = \delta(y, x)$, and $\delta(x, y) \leq \delta(x, z) + \delta(z, y)$, for all words x, y, z, then δ is called a ***distance***. The ***minimum*** difference $\delta(L)$ of a language L containing at least two words is the quantity $\min\{\delta(x, y) : x, y \in L \text{ and } x \neq y\}$. Here we discuss how to compute $\delta\big(\mathrm{L}(\mathbf{a})\big)$, for any given NFA $\mathbf{a}$ accepting at least two words, based on the method of [27]. Questions related to the distance between two languages are studied in [35,13].

Definition 5. *Let δ be an integral difference and let $(\mathbf{t}_i)_{i\in\mathbb{N}}$ be a sequence of transducers. We say that δ is* ***compatible*** *with $(\mathbf{t}_i)_{i\in\mathbb{N}}$, if the following holds, for any $n \in \mathbb{N}$ and $x, y \in \Sigma^*$ with $x \neq y$,*

$$\delta(x, y) \leq n \qquad \text{if and only if} \qquad y \in \mathbf{t}_n(x) \ \text{ or } \ x \in \mathbf{t}_n(y).$$

Example 5. The Hamming distance δ_{H} is compatible with $(\mathtt{sub}_k)_{k\in\mathbb{N}}$ (see Example 1). In [6], the authors consider the general concept of an additive distance and show that many known distances are additive. For example, for two words x, y, with $|x| \geq |y|$, their prefix Hamming distance $\delta_{\mathrm{pH}}(x, y)$ is equal to $\delta_{\mathrm{H}}(x_1, y)+|x_2|$, where $x = x_1x_2$ and $|x_1| = |y|$. This distance is compatible with $(\mathtt{pH}_k)_{k\in\mathbb{N}}$ (see Fig. 2). The results of [21] imply that the edit distance[4] is compatible with a sequence of input-altering transducers—this can be adapted easily to show that also the Levenshtein distance is compatible with a sequence of input-altering transducers. For any two words x, y, their prefix distance, [32], is $|x_1| + |x_2|$, where $x = ux_1$ and $y = uy_1$ and u is the longest common prefix of x, y. It can be shown that this distance also is compatible with a sequence of input-altering transducers similar to $(\mathtt{pH}_k)_{k\in\mathbb{N}}$.

Lemma 1. *Let δ be an integral difference that is compatible with some sequence $(\mathbf{t}_i)_{i\in\mathbb{N}}$ of input-altering transducers, and let L be a language containing at least two words. Then* $\delta(L) = \min\{i \mid \mathbf{t}_i(L) \cap L \neq \emptyset\}$, *assuming* $\min \emptyset = \infty$.

Remark 8. A classic connection between a distance δ and "error"-detection is that a block code C is a k-"error"-detecting if and only if $\delta(C) > k$. Lemma 1 generalizes this connection when we note that L is error-detecting for $(\mathbf{t}_k \vee \mathtt{eq})$ if and only if $\delta(L) > k$.

[4] The ***edit distance*** $\delta_{\mathrm{ed}}(x, y)$ is the minimum number of single-symbol substitutions/insertions/deletions in x that turn x to y. The ***Levenshtein*** distance $\delta_{\mathrm{L}}(x, y)$ is similar by considering only insertions/deletions.

Using Lemma 1, we present below an algorithmic method to compute the minimum difference $m = \delta\big(\mathrm{L}(\mathbf{a})\big)$, for any given NFA $\mathbf{a}$, assuming that (i) δ values are not ∞; (ii) $|\mathrm{L}(\mathbf{a})| \geq 2$; (iii) there is an algorithm computing the input-altering transducer $\mathbf{t}_i$, given any $i \in \mathbb{N}$; (iv) there is an algorithm $\mathrm{UB}_\delta(\mathbf{a})$ computing an upper bound for m, that is, $m \leq \mathrm{UB}_\delta(\mathbf{a})$. In addition to m, the method computes two words in $\mathrm{L}(\mathbf{a})$ whose difference is m. Such words are called ***witnesses*** of minimum difference for $\mathrm{L}(\mathbf{a})$.

$B :=$ the upper bound on $\delta\big(\mathrm{L}(\mathbf{a})\big)$ returned by $\mathrm{UB}_\delta(\mathbf{a})$;
Perform binary search for the smallest $m \in \{1, \ldots, 1 + B\}$ such that
$\big((\mathbf{t}_m \downarrow \mathbf{a}) \uparrow \mathbf{a}\big).\texttt{nEmptyW}() \neq (\texttt{None}, \texttt{None})$
Return m and the word pair $\big((\mathbf{t}_m \downarrow \mathbf{a}) \uparrow \mathbf{a}\big).\texttt{nEmptyW}()$;

This algorithm works in time $O(|\mathbf{a}|^2 M \log B)$, where $M = \max\{|\mathbf{t}_m| \mid m = 1, \ldots, B + 1\}$. Assuming that $\mathrm{UB}_\delta(\mathbf{a})$ works in time $O(|\mathbf{a}|)$ and B is $O(|\mathbf{a}|)$ and M is polynomially bounded in terms of $|\mathbf{a}|$, which is true in many examples of integral differences, we have that also $|\mathbf{a}|^2 M \log B$ is polynomially bounded.

Remark 9. The above algorithmic method is meant to provide a general approach to addressing the minimum difference question as well as an upper bound for the time complexity of it. In some cases however, with a careful algorithm design the time complexity can be improved. For example when δ is the edit distance, [21], the algorithm can be optimized to work in time $O(m \cdot |\mathbf{a}|^2)$, where m is the computed minimum distance of $\mathrm{L}(\mathbf{a})$. Also in [32], the author discusses computing the minimum prefix distance $\delta_{\mathrm{pd}}(L)$ of a regular language L using weighted transducers in time $O(|\mathbf{a}|^2 \log |\mathbf{a}|)$.

Research Directions. It is interesting to investigate connections between additive distances [6] and their compatible transducer sequences. Also interesting is to investigate the use of weighted transducers in computing the minimum difference of a language, for various differences—recall [32] does this for the prefix distance. To this end, it might be necessary to consider the concept of input-altering weighted transducers.

7 Cost of Converting between Transducer Types

Describing an independence with an input-altering transducer, or a difference with compatible input-altering transducers, allows us to answer the satisfaction and minimum difference questions using simple and polynomially bounded algorithms. Moreover, if an independence is described by an input-decreasing transducer then the method of the iterated max-min operator can be used to address the embedding question. Ideally we would like to be able to convert any input-preserving transducer to an input-altering one, or even to an input-decreasing one describing the same language independence. We have the following results.

1. Consider the alphabet $\{0, 1\}$ and the input-preserving transducer $\mathtt{sub}_k$ describing the k-substitution error-detecting property. Then there is an input-decreasing transducer $\mathtt{sub}_k^{\prec}$ describing the same property such that $|\mathtt{sub}_k^{\prec}| = |\mathtt{sub}_k| - 1$.

2. Consider the input-preserving transducer $\mathtt{sid}_k$ describing the k-sid error-detecting property—see Fig. 2 for the case of $k = 2$. In [21], it is shown that there is an input-altering transducer $\mathtt{sid}_k^{\neq}$ describing the same property such that $|\mathtt{sid}_k^{\neq}| = \Theta(|\mathtt{sid}_k|)$.
3. Consider the input-preserving transducer $\mathtt{ov}$ (see Fig. 2) describing the overlap-free property such that no word of an overlap-free language L has a nonempty suffix that is a prefix of another L-word. It can be shown, [28], that there is no input-altering transducer describing the same property.
4. The strictly θ-compliant independence (see Example 2) cannot be described by any input-preserving transducer according to Definition 2, [20].
5. The comma-free code property cannot be described by any transducer (see Remark 10).
6. The transducer $\mathtt{id}_2^{\prec}$ in Fig. 1, [24], is an input-decreasing transducer describing 2-synchronization error-detecting languages over the alphabet $\{0, 1\}$.

Research directions. A natural question here is to investigate the effect on the size of the transducer when converting from an input-altering transducer to an input-decreasing one (when this is possible). For example, converting $\mathtt{id}_k$ to $\mathtt{id}_k^{\prec}$, for any integer $k > 1$. Also interesting is to characterize mathematically when such a conversion is possible for any given input-altering transducer.

8 t-Undescribable Independences and New Directions

Let Σ be the alphabet of independences. In this section we identify some limitations of the transducer method in describing independences (Remark 10), we discuss the method of implicational independence conditions of [16] (Remark 12), and we consider the work of [33] on decision questions related to language equations (Remark 13). These two references provide logical descriptions of language properties. Transducer based methods provide a more operational approach to describing independences, which seems to be a necessary step if one wants to implement objects representing independences [25]. Using all available background information, we explore ways of extending the transducer based method with the aim of describing more independences and at the same time being able to answer related algorithmic questions in a way that some of the algorithms involved can be implemented efficiently in available software systems [11,41].

Remark 10. No n-independence with $n > 3$ is describable by transducers—see Remark 3. The UD-code property is not an n-independence, for any $n \in \mathbb{N}$, [17]. Dependence theory allows us to give simple proofs of such facts. For example, the comma-free code property is not a 3-independence but it is a 4-independence. A language L is a *comma-free* code, [38], if

$$\Sigma^+ L \Sigma^+ \cap LL = \emptyset. \tag{6}$$

To see that this is not a 3-independence, it suffices to consider a 3-word language that is not a comma-free code but any two words of that language constitute a comma-free code. This holds for the language $\{01111, 001011, 0110111\}$.

If A is any language and Op is any union respecting operator (e.g., a transducer operator) then we define the union respecting operators $(\mathrm{Op} \downarrow A)$ and $(\mathrm{Op} \uparrow A)$ such that, for all words v, we have

$$(\mathrm{Op} \downarrow A)(u) = \mathrm{Op}(u \cap A) \quad \text{and} \quad (\mathrm{Op} \uparrow A)(u) = \mathrm{Op}(u) \cap A.$$

Remark 11. Let $\#$ be any symbol not in Σ. Using condition (4) as a guide, we can define the UD-code property as the set of languages L satisfying the condition

$$\text{the operator } \ \mathtt{ins}_{\#} \uparrow (L\#)^+ \ \text{ is functional.} \tag{7}$$

This says that inserting $\#$'s in any word $u \in \Sigma^*$ results in at most one list of L-words where these words are delimited with $\#$.

Remark 12. In [16], the author uses certain first order logic expressions, called implicational independence conditions, to describe independences. That method is aimed for variable-length type of code properties as opposed to typical error-detecting properties of languages such as block codes. The method provides a very general mechanism for expressing independences and answering the satisfaction question. For example, the first condition below describes prefix codes and the second one describes comma-free codes.

$$\forall u, x : \ (u \in L \wedge ux \in L) \ \rightarrow \ (x = \lambda);$$
$$\forall u, u, w, x, y : \ (u \in L \wedge v \in L \wedge w \in L \wedge uv = xwy) \ \rightarrow \ \big((x = \lambda \wedge y = v) \vee (x = u \wedge y = \lambda)\big);$$

where quantification of variables is over a monoid that is typically equal to Σ^*. Quantification over the number of variables is allowed, so as to express independences like the UD-code property, but the syntax for that capability needs to be worked out.

Remark 13. In [33], the author investigates decision questions about language equations. These equations are based on language expressions. There is a set of variables and a set of constants, both representing languages. A ***language expression*** φ is defined inductively as follows: it is a variable, or a constant, or one of $\varphi_1\varphi_2, \varphi_1 \cup \varphi_2, \varphi_1 \cap \varphi_2, \overline{\varphi_1}$, where φ_1 and φ_2 are language expressions. When an equation involves only one variable L, then the set of solutions is a set of languages, which does not necessarily constitute an independence. For example, consider the set $\mathcal{S}$ of solutions of the following language equation

$$L\Sigma^+ \cap \overline{L} = \emptyset. \tag{8}$$

Then, $0\Sigma^*$ satisfies the equation but the subset $\{0\}$ of $0\Sigma^*$ does not satisfy it, so $\mathcal{S}$ cannot be an independence. The cause of this is the presence of the complementation operation $\overline{L}$. On the other hand, the set of solutions of the language equation (6) is exactly the comma-free code property.

Considering that equation (3) describes an independence and considering also conditions (4) and (7), we are led to define independence expressions involving one variable L. An ***independence expression*** φ is defined inductively as follows: it is L, or a language constant, or one of $\varphi_1\varphi_2, \varphi_1 \cup \varphi_2, \varphi_1 \cap \varphi_2, (\varphi)^*, (\varphi)^+$, $\mathbf{t}(\varphi_1), \theta(\varphi_1)$, where φ_1 and φ_2 are independence expressions, θ is an antimorphic permutation constant, and $\mathbf{t}$ is a transducer constant. We assume here that each language constant is written as a regular expression so it represents a regular language. This implies that when the variable L occurring in φ is replaced with a regular language then also φ evaluates to a regular language. We also note that any regular expression, transducer, or permutation involved might contain symbols outside of Σ (recall Σ is the alphabet of the independences being described).

Definition 6. *Let $\mathbf{t}$ be a transducer and let φ, ψ be independence expressions such that at least one of them contains the variable L. A language L is called $(\mathbf{t}, \varphi, \psi)$-**independent** if*

$$\text{the operator } \mathbf{t} \downarrow \varphi(L) \uparrow \psi(L) \text{ is functional.} \tag{9}$$

*The independence $\mathcal{P}_{\mathbf{t},\varphi,\psi}$ described by $\mathbf{t}, \varphi, \psi$ is the set of all $(\mathbf{t}, \varphi, \psi)$-independent languages. Suppose that φ contains L. A language L is called φ-**independent** if*

$$\varphi(L) = \emptyset \tag{10}$$

Remark 14. It can be shown that the concept of $(\mathbf{t}, \varphi, \psi)$-independence in the above definition is well-defined. Moreover, for every independence expression φ containing L, a language L is φ-independent if and only if it is $(\mathtt{all}_\#, \Sigma^*, \varphi \cup \#)$-independent, where $\mathtt{all}_\#$ is any transducer with input and output alphabet $\Sigma \cup \#$ such that $\mathtt{all}_\#(u) = (\Sigma \cup \#)^*$. It can further be shown that the satisfaction question is decidable, where we assume that, in the given expressions φ and/or ψ, any constant language is represented by a regular expression and the language to be tested is given by an NFA or a regular expression.

Example 6. Any independence described by one of the conditions (2), (3), (6) is a φ-independence for some φ. Any independence described by the condition (1) or (7) is a $(\mathbf{t}, \varphi, \psi)$-independence for some $\mathbf{t}, \varphi, \psi$.

Research directions. The independences proposed in Definition 6 should be investigated in various ways. (i) In terms of dependence theory: e.g., when φ describes an independence $\mathcal{P}$, find an n such that $\mathcal{P}$ is in fact an n-independence. (ii) Compare the expressibility of the independence method in Definition 6 to that in Remark 12. (iii) In terms of algorithmic questions about these independences: e.g., (a) define, for given language L and given φ and/or ψ describing some independence $\mathcal{P}$, what the witnesses are for the non-satisfaction of L with respect to $\mathcal{P}$; (b) investigate maximality possibly for restricted forms of φ in (10).

9 Concluding Remarks

We have surveyed applications of transducers in questions related to independent languages. Describing independences by transducers is an operational approach that has led to the implementation of related objects and algorithms [25]. At the end of some sections we proposed directions for future research. In the previous section we proposed a possible extension of the transducer method for describing independences. The method allows one to treat in a uniform way many existing independences as well as to deal with a priory unknown independences. We close with an example of combining three independences.

1. From variable-length codes: UD-code relative to some language M, [7]: $\mathtt{ins}_{\#} \downarrow M \uparrow (L\#)^+$ is functional.
2. From error-control codes: error-detection for $\mathtt{sub}_k$ (see Ex. 1): $\mathtt{sub}_k^{\neq}(L) \cap L = \emptyset$.
3. From DNA codes: θ-compliance (see Ex. 2): $\mathtt{sco}(L) \cap \theta(L) = \emptyset$.

Let Σ' be a primed and disjoint copy of the alphabet Σ and let $\mathtt{cp}$ be a transducer that outputs a primed copy of any given input Σ^*-word. The conjunction of the above independences is described by the condition

$$\big(\mathtt{ins}'_{\#'} \vee \mathtt{all}_{\#}\big) \downarrow \big(\mathtt{cp}M \cup \Sigma^*\big) \uparrow \Big(\big(\mathtt{cp}(L\#)\big)^+ \cup \big(\varphi(L) \cup \#\big)\Big) \text{ is functional,}$$

where $\varphi(L) = \big(\mathtt{sub}_k^{\neq}(L) \cap L\big) \cup \big(\mathtt{sco}(L) \cap \theta(L)\big)$, and $\mathtt{ins}'_{\#'}$ is the primed version of $\mathtt{ins}_{\#}$.

References

1. Allauzen, C., Mohri, M.: Efficient algorithms for testing the twins property. J. Automata, Languages and Combinatorics 8(2), 117–144 (2003)
2. Béal, M.P., Carton, O., Prieur, C., Sakarovitch, J.: Squaring transducers: An efficient procedure for deciding functionality and sequentiality. Theoretical Computer Science 292(1), 45–63 (2003)
3. Berstel, J.: Transductions and Context-Free Languages. B.G. Teubner, Stuttgart (1979)
4. Berstel, J., Perrin, D., Reutenauer, C.: Codes and Automata. Cambridge University Press (2009)
5. Bruyère, V.: Maximal codes with bounded deciphering delay. Theoretical Computer Science 84, 53–76 (1991)
6. Calude, C., Salomaa, K., Yu, S.: Additive distances and quasi-distances between words. Journal Of Universal Computer Science 8:2, 141–152 (2002)
7. Daley, M., Jürgensen, H., Kari, L., Mahalingam, K.: Relativized codes. Theoretical Computer Science 429, 54–64 (2012)
8. Domaratzki, M.: Trajectory-based codes. Acta Informatica 40, 491–527 (2004)
9. Dudzinski, K., Konstantinidis, S.: Formal descriptions of code properties: decidability, complexity, implementation. International Journal of Foundations of Computer Science 23:1, 67–85 (2012)
10. Ehrenfeucht, A., Rozenberg, G.: Each regular code is included in a maximal regular code. RAIRO Inform. Théor. Appl. 20, 89–96 (1985)

11. FAdo: Tools for formal languages manipulation, URL address: http://fado.dcc.fc.up.pt/ Accessed in April, 2017
12. Hamming, R.W.: Error detecting and error correcting codes. The Bell System Technical Journal 26(2), 147–160 (1950)
13. Han, Y.S., Ko, S.K., Salomaa, K.: Computing the edit-distance between a regular language and a context-free langauge. In: Yen, H.C., Ibarra, O. (eds.) DLT 2012. LNCS 7410. pp. 85–96. Springer, Heidelberg (2012)
14. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley (1979)
15. Jonoska, N., Mahalingam, K.: Languages of DNA based code words. In: Proceedings of 9th Intern. Workshop on DNA-Based Computers, DNA 9, Madison, Wisconsin, 2003. pp. 58–68. No. 2943 in LNCS (2004)
16. Jürgensen, H.: Syntactic monoids of codes. Acta Cybernetica 14, 117–133 (1999)
17. Jürgensen, H., Konstantinidis, S.: Codes. In: Rozenberg and Salomaa [36], pp. 511–607
18. Kamabe, H.: Outfix-free and intercode constraints for DNA sequences. In: Proceedings of 2011 IEEE Intern. Symposium on Inform. Theory. pp. 1574–1578 (2011)
19. Kari, L., Kitto, R., Thierrin, G.: Codes, involutions, and DNA encodings. In: Formal and Natural Computing - Essays Dedicated to Grzegorz Rozenberg. pp. 376–393 (2002)
20. Kari, L., Konstantinidis, S., Kopecki, S.: Transducer descriptions of DNA code properties and undecidability of antimorphic problems. Information and Computation (2017), to appear—journal version of paper in LNCS 9118, Proceedings of DCFS 2015, Waterloo, Canada, pp 141–152.
21. Kari, L., Konstantinidis, S., Kopecki, S., Yang, M.: An efficient algorithm for computing the edit distance of a regular language via input-altering transducers. CoRR abs/1406.1041 (2014), http://arxiv.org/abs/1406.1041
22. Kari, L., Konstantinidis, S., Sosík, P.: On properties of bond-free DNA languages. Theoretical Computer Science 334, 131–159 (2005)
23. Konstantinidis, S.: Transducers and the properties of error-detection, error-correction and finite-delay decodability. Journal Of Universal Computer Science 8, 278–291 (2002)
24. Konstantinidis, S., Mastnak, M.: Embedding rationally independent languages into maximal ones. J. Automata, Languages and Combinatorics (2017), to appear
25. Konstantinidis, S., Meijer, C., Moreira, N., Reis, R.: Implementation of code properties via transducers. In: Han, Y.S., Salomaa, K. (eds.) Proceedings of CIAA 2016. pp. 1–13. No. 9705 in Lecture Notes in Computer Science (2016)
26. Konstantinidis, S., Moreira, N., Reis, R.: Generating error control codes with automata and transducers. In: Bordihn, H., Freund, R., Nagy, B., Vaszil, G. (eds.) Proceedings of NCMA 2016. pp. 211–226. No. 321 in Österreichische Computer Gesellschaft (2016)
27. Konstantinidis, S., Silva, P.V.: Computing maximal error-detecting capabilities and distances of regular languages. Fundamenta Informaticae 101(4), 257–270 (2010)
28. Kopecki, S.: Personal communication (2013)
29. Lam, N.H.: Finite maximal solid codes. Theoretical Computer Science 262, 333–347 (2001)
30. Mateescu, A., Salomaa, A.: Formal languages: an introduction and a synopsis. In: Rozenberg and Salomaa [36], pp. 1–39
31. Mercier, H., Bhargava, V.K., Tarokh, V.: A survey of error-correcting codes for channels with symbol synchronization errors. IEEE Communications Surveys & Tutorials 12, 87–96 (2010)

32. Ng, T.: Prefix distance between regular languages. In: Han, Y.S., Salomaa, K. (eds.) Proceedings of 21st CIAA, July 19–22, 2016, Seoul, South Korea. Lecture Notes in Computer Science, vol. 9705, pp. 224–235 (2016)
33. Okhotin, A.: Decision problems for language equations. Journal of Computer and System Sciences 76, 251–266 (2010)
34. Paluncic, F., Abdel-Ghaffar, K., Ferreira, H.: Insertion/deletion detecting codes and the boundary problem. IEEE Trans. Information Theory 59(9), 5935–5943 (2013)
35. Pighizzini, G.: How hard is computing the edit distance? Information and Computation 165, 1–13 (2001)
36. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages, Vol. I. Springer-Verlag, Berlin (1997)
37. Sakarovitch, J.: Elements of Automata Theory. Cambridge University Press, Berlin (2009)
38. Shyr, H.J.: Free Monoids and Languages. Hon Min Book Company, Taichung, second edn. (1991)
39. Shyr, H.J., Thierrin, G.: Codes and binary relations. In: Malliavin, M.P. (ed.) Séminaire d'Algèbre Paul Dubreil, Paris 1975–1976 (29ème Année). Lecture Notes in Mathematics, vol. 586, pp. 180–188 (1977)
40. Van, D.L., Hung, K.V., Huy, P.T.: Codes and length-increasing transitive binary relations. In: Hung, D., Wirsing, M. (eds.) Proceedings of ICTAC 2005. Lecture Notes in Computer Science, vol. 3722, pp. 29–48 (2005)
41. Vaucanson: The vaucanson project, URL address: `http://vaucanson-project.org/` Accessed in April, 2017
42. Wood, D.: Theory of Computation. Harper & Row, New York (1987)
43. Yu, S.: Regular languages. In: Rozenberg and Salomaa [36], pp. 41–110
44. Yu, S.S.: Languages and codes. Tsang Hai Book Publishing, Taichung (2005)
45. Zaccagnino, R., Zizza, R., Zottoli, C.: Testing DNA code words properties of regular languages. Theoretical Computer Science 608, 84–97 (2015)
46. Zhang, L., Shen, Z.: Completion of recognizable bifix codes. Theoretical computer science 145, 345–355 (1995)

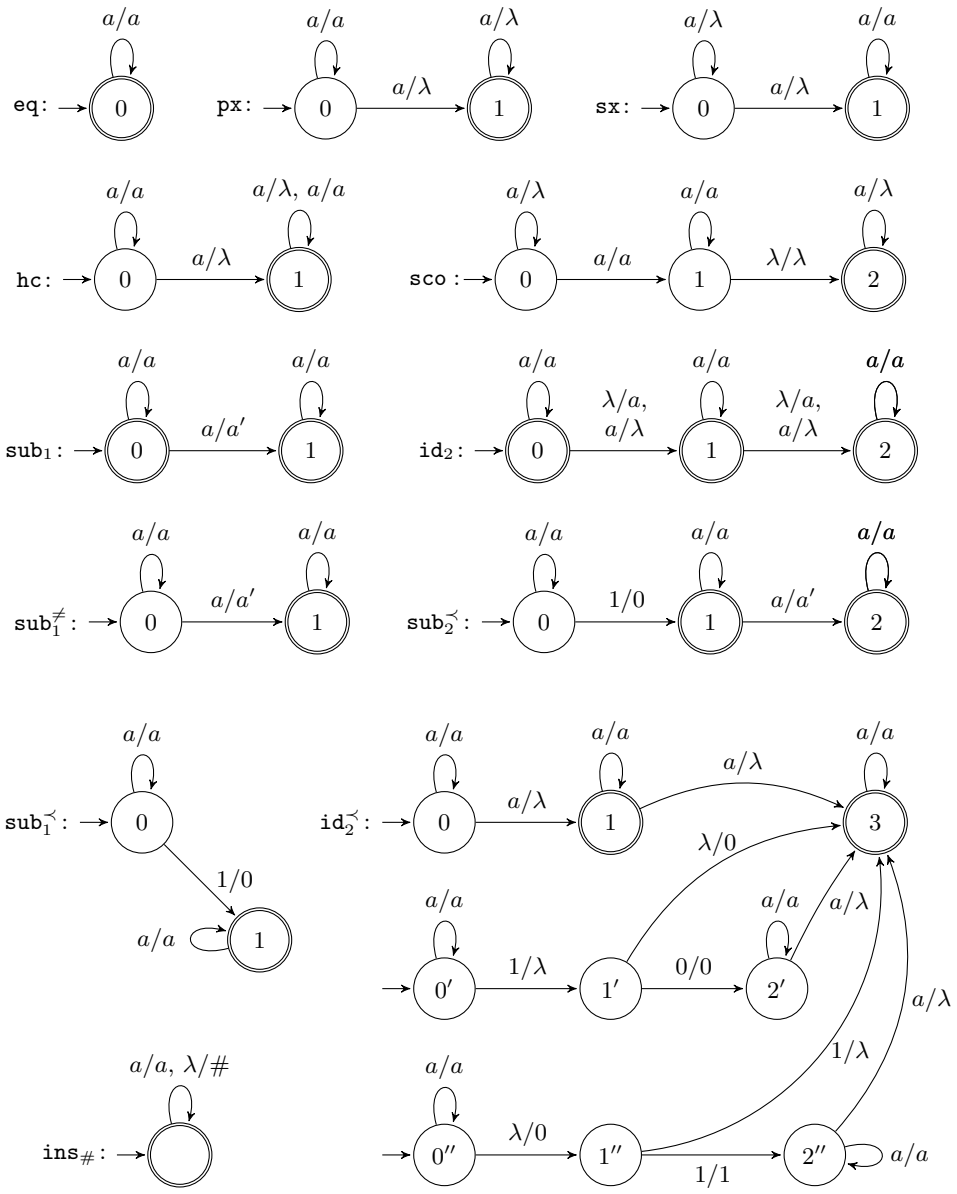

Fig. 1. Various transducers. An arrow with label a/a denotes multiple transitions: one with label a/a for each $a \in \Sigma$, and similarly for labels a/λ. An arrow with label a/a' denotes multiple transitions: one with label a/a' for all $a, a' \in \Sigma$ with $a \neq a'$. Let w be any word. We have: $\texttt{eq}(w) = \{w\}$; $\texttt{px}(w) =$ set of proper prefixes of w; $\texttt{sx}(w) =$ set of proper suffixes of w; $\texttt{hc}(w) =$ set of words resulting by deleting at least one symbol in w; $\texttt{sub}_1^{\prec}(w) =$ set of words resulting by substituting exactly one 1 in w with a 0; $\texttt{sub}_1(w) =$ set of words resulting by substituting at most one symbol in w with another one; $\texttt{id}_2(w)$ = set of words resulting by inserting and/or deleting at most 2 symbols in w.

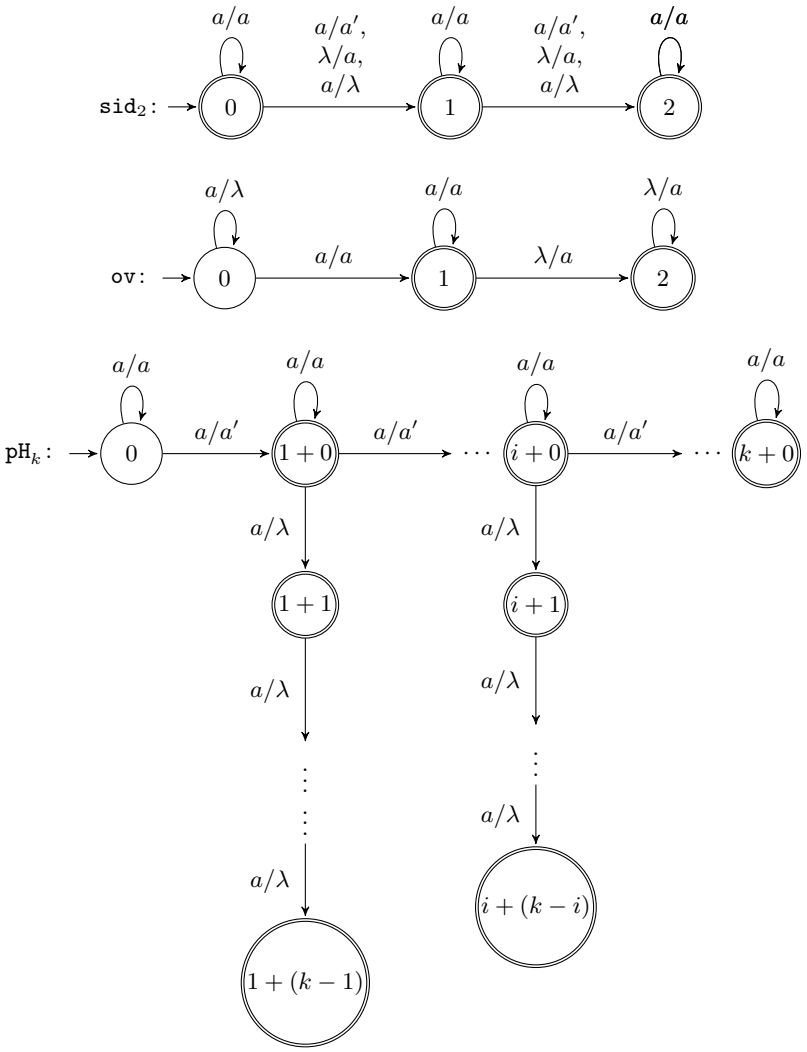

Fig. 2. More transducers. Let w be any word. We have: $\texttt{sid}_2(w)$ = set of words resulting by substituting and/or inserting and/or deleting a total of at most 2 symbols in w; $\texttt{ov}(w)$ = set of words (including w) having a nonempty prefix that is equal to a suffix of w; $\texttt{pH}_k(w)$ = set of words $x \neq w$ such that $|w| \geq |x|$ and the prefix Hamming distance of x, w is at most k—see Example 5.