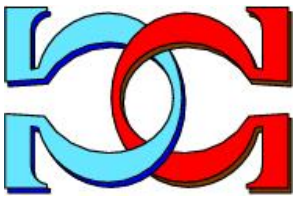
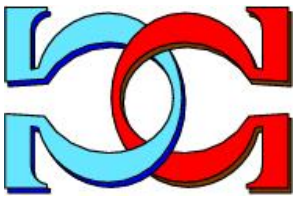
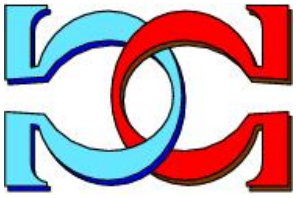
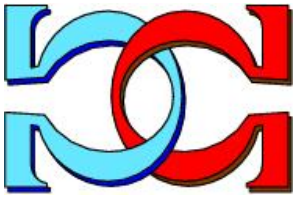


CDMTCS
Research
Report
Series

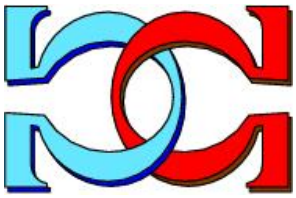




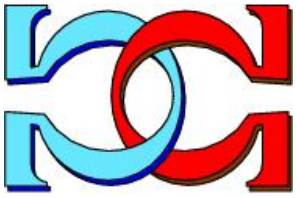
Automata for Solid Codes



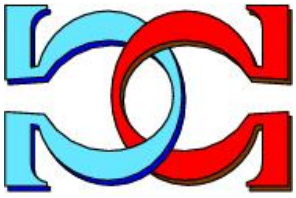
Helmut Jürgensen
The University of Western Ontario,
London, Ontario



Ludwig Staiger
Martin-Luther-Universität
Halle-Wittenberg



CDMTCS-550
January 2021



Centre for Discrete Mathematics and
Theoretical Computer Science

Automata for Solid Codes

Helmut Jürgensen*

and

Ludwig Staiger†
Martin-Luther-Universität Halle-Wittenberg
Institut für Informatik
von-Seckendorff-Platz 1, D–06099 Halle (Saale), Germany

Abstract

Solid codes provide outstanding fault-tolerance when used for information transmission through a noisy channel involving not only symbol substitutions, but also synchronisation errors and black-outs. In this paper we provide an automaton theoretic characterisation of solid codes which takes this fault-tolerance into account.

The fault-tolerance afforded by a solid code L can be summarised as follows: Consider messages, encoded using L, being sent through a noisy channel. Any code words in L, which are present in the received message, will be decoded correctly, unless they themselves happen to be the results of errors. Thus, errors in the received message will not lead to incorrect decodings of those parts which are error-free.

In this paper we consider acceptors which are fault-tolerant in this sense when analysing such received messages. These acceptors characterise the class of solid codes. For finite solid codes an automaton characterisation was published in the sixties by Levenshtein and Romanov. The characterisation uses state-invariant finite-state transducers which act as decoders in such a way that an output is generated exactly when a code word has been read

*Helmut Jürgensen was with the Department of Computer Science, The University of Western Ontario, London, Ontario, Canada

†email: `ludwig.staiger@informatik.uni-halle.de`

completely. State-invariance means that acceptance does not depend on the initial state - every state can be used as the initial state.

The results of Levenshtein and Romanov depend strongly on the fact that the code is finite. In this paper we provide a general automaton theoretic characterisation of arbitrary solid codes without any such restriction. Moreover, the solid code is regular as a language if and only if the automaton used in the characterisation can be reduced to an equivalent finite automaton with equivalent properties.

The main results of this paper are as follows: Every acceptor defines a solid code. For every solid code there is a fault-tolerant acceptor defining the code. Such acceptors expose the decomposition of potentially faulty received messages according to the code. For solid codes which are regular as languages these acceptors can be chosen to be finite while preserving all important combinatorial properties.

Contents

1 Automata for Codes

Automaton theory has many rôles in coding theory: automata are used to characterise classes of codes; they are employed to decide code properties; transducers serve as abstract models for encoding and decoding; they express error resistance properties; they can be used as filters for pattern matching; etc. Many details about automaton theoretic aspects of coding theory, but not all, can be found in the book "Codes and Automata" by Berstel, Perrin and Reutenauer [BPR10]. The critical surveys [Jür13, Jür14] may add further perspectives.

When considering the theory of codes, one has to distinguish two nearly disjoint fields, both called "coding theory": the theory of error-correcting codes, mainly block codes (see [HB98], for example); the theory of codes with words of possibly different lengths, sometimes referred to as variable-length codes (see [BP85, Shy01], for example). A systematic proposal to unify these different aspects was made in [JK97].

Automaton theoretic models are of particular interest for codes which are finite or regular languages, because the corresponding algorithms rely on a finite-state transition processes. In many cases the restriction to finitely many states is not essential, but provides a deeper understanding of the computations involved and makes several properties decidable [JK97, FRS07].

Automaton theoretic studies of codes take into account at least the fol- lowing different points of view, when considering a given class $\mathcal{L}$ of languages $L \subseteq X^*$ satisfying certain properties.

- Acceptors for exactly the languages $L \in \mathcal{L}$.
- Acceptors or decoders for exactly the messages encoded by languages $L \in \mathcal{L}$, that is for L^* with $L \in \mathcal{L}$.
- Fault-tolerant acceptors or decoders for the messages encoded by languages $L \in \mathcal{L}$.
- Error-correcting acceptors or decoders for the messages encoded by languages $L \in \mathcal{L}$.

Obviously, these settings are quite different. In this paper we describe fault-tolerant acceptors for messages encoded by solid codes.

For finite solid codes such automata were characterised in 1964 as state-invariant decoders without look-ahead (see Section 5). This characterisation is due to Levenshtein [Lev64a] and Romanov [Rom66]. We

generalise these considerations to infinite solid codes and show how a finite description of such codes translates into a finite description of a finite fault-tolerant acceptor for such codes. Intuitively, these acceptors are also state-invariant and do not resort to look-ahead. In general, to handle such situations, one needs to refer to a channel model as outlined in [JK97].

We show that every acceptor defines a solid code and that every solid code can be represented in this way. The acceptors considered in this context may be infinite; they are fault tolerant in the following sense: in an encoded message received after transmission through a noisy channel they correctly identify those code words which have been transmitted correctly. Thus, we do not look at acceptors of solid codes themselves, but at fault-tolerant acceptors of messages encoded by solid codes.

Consider a message encoded using a solid code. The encoded message is a sequence of code words. The message received after transmission through a noisy channel is a sequence of symbols the combination of some of which may result in the recovery of a code word. The usage of a solid code guarantees, that code words in the received message are detected unequivocally. A fault-tolerant acceptor, when reading a received message, will indicate the appearance of code words and possibly enable a decoding.

One of the best known characterisations of a class of codes by automata concerns prefix codes: Prefix codes are precisely the languages accepted by tree-shaped automata with the initial state as the root and the leaves as accepting states. Alternatively, if A is an acceptor with input alphabet X and $L(A)$ is the language accepted by A, then the language $L(A) \setminus L(A)X^+$ is a prefix code or consists only of the empty word. Moreover, every prefix code is obtained in this fashion. When A is finite then the prefix code is regular.

Similar, but less explicit characterisations exist also for hypercodes [Val77, Thi73, Thi81] and code classes related to infix or outfix codes [IJST91, Jür99, PT90], typically in terms of the syntactic monoid of such codes. In addition, many studies concern the automaton theoretic characterisation of the set of messages encoded using a given code. To get a fairly comprehensive understanding of the situation we refer to [BPR10, JK97, Shy01, Yu05]. Further important early studies concerning the connection between automata and codes include [Gle61, Lev61, Lev62, Lev64b, Mar62].

In this paper we consider the class of solid codes. A solid code is a language L such that every word has a unique decomposition into

words belonging to L and words which do not contain a word in L. Thus, solid codes are strongly resistant to transmission errors: those parts of a message which were transmitted correctly are also decoded correctly. We provide further details about solid codes in Section 4 below.

For finite solid codes an automaton characterisation was given by Levenshtein [Lev64a] and Romanov [Rom66]. The characterisation uses state-invariant finite-state transducers which act as decoders in such a way that an output is generated exactly when a code word has been read completely. State-invariance means that the acceptance does not depend on the initial state – every state can be used as the initial state. Intuitively, state-invariance expresses the fact that, when an encoded message is read after transmission through a noisy channel, the decoder will find the code words regardless of intermediate incorrect parts.

The results of Levenshtein and Romanov depend strongly on the fact that the code is finite. In this paper we provide a general automaton theoretic characterisation of arbitrary solid codes. Moreover, the solid code is regular as a language if and only if the automaton used in the characterisation can be reduced to an equivalent finite automaton. Our characterisation does not involve decoders, but only acceptors. To define decoders one would need to know how the encoding is done: for a finite code L one can make the simple assumption that L encodes the symbols in an alphabet of size $|L|$; for an infinite code L the encoding would be specified by significantly more complicated transducer, the choice of which depends very much on the technical circumstances.

We use ideas from the constructions of Levenshtein and Romanov. Intuitively, though not quite truly, the acceptors involved are state-invariant. The concept of a decoded output being issued exactly at the end of reading a code word is modelled by accepting a word only if no accepting state was entered before the end of the word.

The main results of this paper are as follows: Every acceptor defines a solid code. For every solid code there is a fault-tolerant acceptor defining the code. Such acceptors expose the decomposition of potentially faulty received messages according to the code. For solid codes which are regular as languages these acceptors can be chosen to be finite while preserving all important combinatorial properties.

Our paper is structured as follows: In Section 2 we introduce the notation and some basic notions. Properties of solid codes are reviewed in Section 4. In Section 5 we define state-invariant decoders without look-ahead for finite solid codes and summarise the corresponding results of Levenshtein and Romanov. We introduce the main tool for our con-

struction, the Levenshtein-Romanov mapping, in Section 6. We prove several of its properties under various assumptions. This leads to new characterisations of several classes of codes in s Section 7. In Section 8 we show that every acceptor defines a solid code and that every solid code defines a fault-tolerant acceptor. However, the fault-tolerant acceptors defined by solid codes are usually infinite, even when the solid code is a regular language. In Section 9 we show how to construct a reduced fault-tolerant acceptor for a solid code.

2 Notation and Basic Notions

We introduce some notation and review several notions and facts needed in this paper. As general references we use: the "Handbook of Formal Languages" [RS97a, RS97b] for the theories of formal languages and automata; the monographs "Abstrakte Automaten" (Abstract Automata) [Sta69] and "Algebraic Theory of Automata" [GP72] for foundations of automaton theory; the monographs "Theory of Codes" [BP85], "Codes and Automata" [BPR10], "Free Monoids and Languages" [Shy01], "Languages and Codes" [Yu05] and the article "Codes" [JK97] for the theory of codes.

By $\mathbb{N}$ and $\mathbb{N}_0$ we denote the sets of positive and non-negative integers, respectively. We use the common notation for operations on sets; when there is no risk of confusion, we omit set brackets for singleton sets.

Let X be a set, and let $\cong$ be an equivalence relation on X. For $x \in X$, $[x]_\cong$ is the equivalence class of x, and $X/\cong$ is the factor set, that is the set of equivalence classes of the elements of X. We write $[x]$ instead of $[x]_\cong$ when which equivalence relation is used is obvious from the context.

An alphabet is a non-empty set. The elements of an alphabet are called symbols. Finite sequences of symbols are called words.

Let X be an alphabet. The set of all words over X, including the empty word ε, is denoted by X^*. To exclude the empty word we write X^+, that is, $X^+ = X^* \setminus \{\varepsilon\}$. With concatenation of words as multiplication, the set X^* is a monoid. A language over X is a subset of X^*. For a language $L \subseteq X^*$ and a word $u \in X^*$, the set

$$u^{co[-1]}L = \{w | w \in X^*, uw \in L\}$$

is the left quotient of L by u. Given a language L, two words $u, v \in X^*$ are said to be Nerode equivalent with respect to L, $u \equiv_L v$, if and only if

$u^{[-1]}L = v^{[-1]}L$. Then $[u]_L$ denotes the Nerode equivalence class of u.

In our context, the case when the alphabet X is a singleton set usually leads to trivial and clumsy exceptions. In the sequel we assume, unless stated otherwise, that an alphabet contains at least two distinct symbols and these will include a and b or other ones as needed without special mention.

For a word $w \in X^*$,

$$\mathsf{Pref}(w) = \{u | u \in X^*, w \in uX^*\}$$

is the set of prefixes of w. The set of proper prefixes of w is

$$\mathsf{Pref}_+(w) = \mathsf{Pref}(w) \setminus \{\varepsilon, w\}.$$

For $L \subseteq X^*$, let

$$\mathsf{Pref}(L) = \bigcup_{w \in L} \mathsf{Pref}(w) \text{ and } \mathsf{Pref}_+(L) = \bigcup_{w \in L} \mathsf{Pref}_+(w).$$

Note that $\varepsilon \notin \mathsf{Pref}_+(L)$, but that $L \cap \mathsf{Pref}_+(L) = \emptyset$ is possible. One defines suffixes and infixes of words analogously. Thus

$$\mathsf{Suff}(w) = \{u | u \in X^*, w \in X^*u\}$$

is the set of suffixes of w, and

$$\mathsf{Inf}(w) = \{u | u \in X^*, w \in X^*uX^*\}$$

is the set of infixes of w. The sets of proper suffixes and proper infixes of w are

$$\mathsf{Suff}_+(w) = \mathsf{Suff}(w) \setminus \{\varepsilon, w\}$$

and

$$\mathsf{Inf}_+(w) = \mathsf{Inf}(w) \setminus \{\varepsilon, w\},$$

respectively. For $r \in \{\mathsf{Pref}, \mathsf{Suff}\}$ and $L \subseteq X^+$, the *r-root* of L is the set

$$\sqrt[r]{L} = \{w | w \in L, r(w) \cap L = \emptyset\}.$$

As $\varepsilon \notin L$, one has $\varepsilon \notin \sqrt[r]{L}$.

Let $u, v \in X^*$. The *shuffle product* of u and v is the set

$$u \sqcup\!\sqcup v = \left\{ w \;\middle|\; \begin{array}{l} w \in X^*, \exists n \in \mathbb{N} \exists u_0, \ldots, u_n \in X^* \\ \exists v_1, \ldots, v_n \in X^* : u = u_0 u_1 \cdots u_n, \\ v = v_1 v_2 \cdots v_n, w = u_0 v_1 u_1 v_2 u_2 \cdots v_n u_n \end{array} \right\}$$

For languages $L, L' \subseteq X^*$, the shuffle product of L and L' is the set

$$L \sqcup\!\sqcup L' = \bigcup_{u \in L, v \in L'} u \sqcup\!\sqcup v \,.$$

We consider the following classes of codes or languages $L \subseteq X^+$ related to codes:

NAME	DEFINITION	CLASS
prefix code	$LX^+ \cap L = \emptyset$	$\mathcal{L}_{\mathrm{p}}$
suffix code	$X^+L \cap L = \emptyset$	$\mathcal{L}_{\mathrm{s}}$
infix code	$\mathsf{Inf}_+(L) \cap L = \emptyset$	$\mathcal{L}_{\mathrm{i}}$
bifix code	$L \in \mathcal{L}_{\mathrm{p}} \cap \mathcal{L}_{\mathrm{s}}$	$\mathcal{L}_{\mathrm{b}}$
overlap-free[1]	$\mathsf{Pref}_+(L) \cap \mathsf{Suff}_+(L) = \emptyset$	$\mathcal{L}_{\mathrm{ol}}$
solid code	$L \in \mathcal{L}_{\mathrm{i}} \cap \mathcal{L}_{\mathrm{ol}}$	$\mathcal{L}_{\mathrm{solid}}$
p-infix code	$X^*LX^+ \cap L = \emptyset$	$\mathcal{L}_{\mathrm{pi}}$
s-infix code	$X^+LX^* \cap L = \emptyset$	$\mathcal{L}_{\mathrm{si}}$
comma-free code	$X^+LX^+ \cap L^2 = \emptyset$	$\mathcal{L}_{\mathrm{comma-free}}$
hypercode	$(L \sqcup\!\sqcup X^+) \cap L = \emptyset$	$\mathcal{L}_{\mathrm{h}}$

To keep statements simple, we allow for L to be empty in all these cases. A language $L \subseteq X^+$ is a prefix code or a suffix code, if and only if $\sqrt[\mathsf{Pref}]{L} = L$ or $\sqrt[\mathsf{Suff}]{L} = L$, respectively. For details about the classes of languages, see [JK97]. We summarise the relations between these classes:

- $\mathcal{L}_{\mathrm{solid}} = \mathcal{L}_{\mathrm{i}} \cap \mathcal{L}_{\mathrm{ol}} \subset \mathcal{L}_{\mathrm{comma-free}} \subset \mathcal{L}_{\mathrm{i}} = \mathcal{L}_{\mathrm{pi}} \cap \mathcal{L}_{\mathrm{si}}$;
- $\mathcal{L}_{\mathrm{i}} \subset \mathcal{L}_{\mathrm{pi}} \subset \mathcal{L}_{\mathrm{p}}$;
- $\mathcal{L}_{\mathrm{i}} \subset \mathcal{L}_{\mathrm{si}} \subset \mathcal{L}_{\mathrm{s}}$;
- $\mathcal{L}_{\mathrm{i}} \subset \mathcal{L}_{\mathrm{b}} = \mathcal{L}_{\mathrm{p}} \cap \mathcal{L}_{\mathrm{s}}$;
- $\mathcal{L}_{\mathrm{h}} \cap \mathcal{L}_{\mathrm{solid}} \subset \mathcal{L}_{\mathrm{h}} \subset \mathcal{L}_{\mathrm{i}}$.

The containments are shown in Fig. 1.

Most of the usual classes of codes can be described conveniently as independent sets for various types of dependence systems on X^* or as sets of incomparable words with respect to certain n-ary relations on X^*. For details we refer to [JK97] and the literature cited there. In this paper we refer to only a small number of such relations, all binary, and all being partial orders on X^*. Their list is as follows for $u, v \in X^*$:

[1] Observe that overlap-free languages are not necessarily codes, e.g. $\{a, ab, aab\}$.

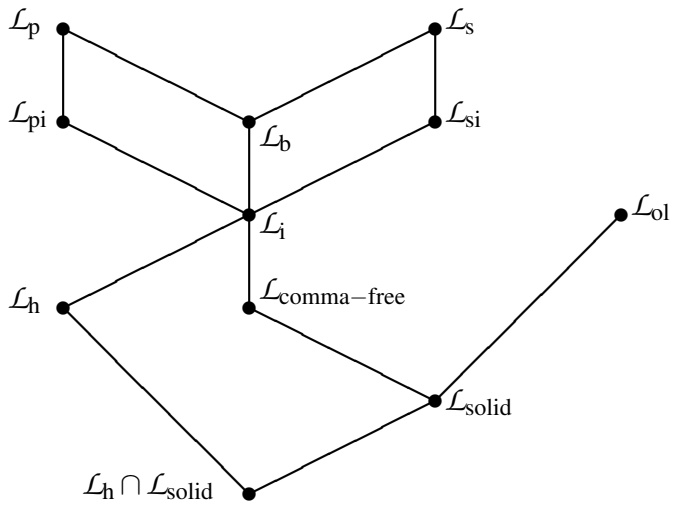


Figure 1: Relations between the language or code classes considered.

RELATION	DEFINITION	INDEPENDENT LANGUAGES
$u \leq_{\mathrm{p}} v$	$v \in uX^*$	prefix codes
$u \leq_{\mathrm{s}} v$	$v \in X^*u$	suffix codes
$u \leq_{\mathrm{i}} v$	$v \in X^*uX^*$	infix codes

The strict versions of these partial orders require that $u \neq v$. Thus, for example, one has $u <_{\mathrm{p}} v$ if and only if $v \in uX^+$. We also use the length-lexicographical order $\leq_{\mathrm{ll}}$ on X^*. To define this order one assumes an arbitrary, but fixed total order $\leq_{\mathrm{alph}}$ on X, which, in turn, defines the lexicographical order $\leq_{\mathrm{lex}}$ on X^*. Then the length-lexicographical order is defined by the following conditions: $u \leq_{\mathrm{ll}} v$ if and only if

1. either $|u| < |v|$
2. or $|u| = |v|$ and $u \leq_{\mathrm{lex}} v$.

The length-lexicographical order is not interesting from the point of view of codes. Its sets of incomparable words are singleton sets. It is, however, useful for the enumeration of words in X^*.

We now turn to basic definitions in automaton theory. We do not make any assumptions about finiteness or computability. Such assumptions are introduced only when needed.

Definition 1 A *deterministic semi-automaton* is a construct $A = (Q, X, \delta)$ such that

1. Q is a non-empty set, the set of states;

2. X is a finite alphabet, the input alphabet;

3. $\delta : Q \times X \to Q$ is the transition function.

We permit the set of states to be infinite; we also permit the transition function to be non-computable. The input alphabet is always assumed to be finite. As we do not consider non-deterministic semi-automata in this paper, we omit the word 'deterministic' in the sequel. A semi-automaton is said to be finite if its set of states is finite. The definition of δ is extended to $Q \times X^* \to Q$ by sequentiality as usual.

Definition 2 A *deterministic acceptor* is a construct $A = (Q, X, \delta, q_0, F)$ such that

1. (Q, X, δ) is a deterministic semi-automaton;

2. $q_0 \in Q$ is the initial state;

3. $F \subseteq Q$ is the set of accepting states.

The *language accepted by* an acceptor A as above is the set

$$L(A) = \{w | w \in X^*, \delta(q_0, w) \in F\}.$$

A language is *regular* (or *rational*) if and only if it is accepted by a finite acceptor. Occasionally we also need to consider further classes in the language hierarchy; for those we refer to the general references listed above.

An acceptor is *(initially) connected* if, for every state $q \in Q$, there is an input word $w \in X^*$ such that $\delta(q_0, w) = q$. It is *strongly connected* if, for any two states $q, q' \in Q$, there is an input word $w \in X^*$ such that $\delta(q, w) = q$. A state $q \in Q$ is said to be useless if, for every word $w \in X^*$, $\delta(q, w) \notin F$.

There are several instances when we need to change the initial state of an acceptor. To indicate the fact that q_0 has been replaced by state q as the initial state we write A_q. If q is a useless state, then $L(A_q) = \emptyset$. An acceptor, which is initially connected and has no useless states, need not be strongly connected. On the other hand, if A is strongly connected and $F \neq \emptyset$, then A has no useless states and is, a fortiori, initially connected.

Occasionally we need to consider an acceptor, the transition function of which is only a partial function. Such an acceptor is said to be partial; we sometimes emphasise the fact that the acceptor is not partial by saying that it is a total acceptor.

Let $A = (Q, X, \delta, q_0, F)$ and $A' = (Q', X, \delta', q'_0, F')$ be two partial or total acceptors. A is said to be a *(partial) subacceptor* of A if the following four conditions are satisfied:

1. $Q' \subseteq Q$;
2. $q'_0 = q_0$;
3. $F' = Q' \cap F$;
4. for all $q \in Q'$ and all $x \in X$, if $\delta'(q,x)$ is defined, then also $\delta(q,x)$ is defined and $\delta'(q,x) = \delta(q,x)$.

In such a case we write $A' \preceq A$.

Definition 3 Let $A = (Q, X, \delta, q_0, F)$ be an acceptor, and let $i \in \mathbb{N}$. A word $w \in X^+$ is *accepted by A at stage i* if the following conditions are met:

1. $\delta(q_0, w) \in F$;
2. there are exactly $i-1$ distinct prefixes $w_1, w_2, \ldots, w_{i-1} \in \mathsf{Pref}_+(w)$ such that $\delta(q_0, w_j) \in F$ for $j = 1, 2, \ldots, i-1$.

We denote by $L_i(A)$ the set of words which are accepted by A at stage i.

We list a few immediate consequences of this definition:

1. $L_1(A)$ is the set of all non-empty words w such that $\delta(q_0, w) \in F$ with $\delta(q_0, u) \notin F$ for every proper prefix u of w. In particular, when $q_0 \in F$, this excludes the empty word. See Fig. 2.
2. $L_i(A) \cap L_j(A) = \emptyset$ for $i \neq j$.
3. The set $\bigcup_{i=1}^{\infty} L_i(A)$ is the set of all non-empty words accepted by A, that is, $\bigcup_{i=1}^{\infty} L_i(A) = L(A) \setminus \{\varepsilon\}$.

Remark 1 *The definition of $L_i(A)$ does not depend on the acceptor A. Consider a non-empty language $L \subseteq X^+$. For $i \in \mathbb{N}$, let L_i be the set of words in L which have exactly $i-1$ proper prefixes in L. Let A be any deterministic acceptor[2] for L, that is, $L = L(A)$. Then $L_i(A) = L_i$.*

Recall that an equivalence relation $\cong$ on the set Q of states of an acceptor $A = (Q, X, \delta, q_0, F)$ is a *congruence* if and only if the following conditions are satisfied for all $q, q' \in Q$ and all $x \in X$:

[2]Such an acceptor always exists. It can, for instance, be obtained by the Nerode construction defined further below.

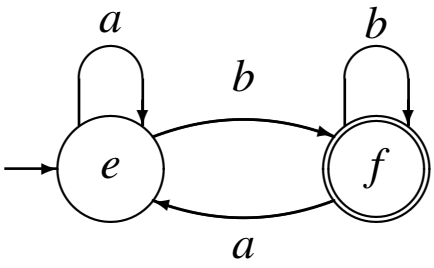


Figure 2: An automaton illustrating the acceptance at stages. There is only one accepting state, f . One has $L_1(A_e) = a^*b$ and $L_1(A_f) = \{b\} \cup a^+b = a^*b$. In general, for $i > 0$, $L_i(A_e) = L_i(A_f) = (a^*b)^i$.

1. If $q' \cong q$ then q' and q are either both in F or both in $Q \setminus F$.

2. If $q' \cong q$ then $\delta(q',x) \cong \delta(q,x)$.

Given a congruence $\cong$, the factor acceptor is defined as

$$A/\cong = (Q/\cong, X, \delta_\cong, [q_0]_\cong, F/\cong)$$

where

$$\delta_\cong([q]_\cong, x) = [\delta(q,x)]_\cong.$$

The following remark is used several times in the sequel.

Remark 2 *Let A be an acceptor and let $\cong$ be a congruence on A. One has $L(A) = L(A/\cong)$. Moreover, using Remark 1, one has $L_i(A) = L_i(A/\cong)$ for all $i \in \mathbb{N}$.*

Each acceptor has a unique maximal congruence $\cong_{\max}$, often referred to as *state equivalence*: $q', q \in Q$ are equivalent, $q' \cong_{\max} q$, if and only if $L(A_{q'}) = L(A_q)$. The acceptor $A/\cong_{\max}$ is the reduced acceptor for A. If $A/\cong_{\max}$ is finite, then it has the smallest number of states among all acceptors for the language $L(A)$. See [Sta69, GP72] or other advanced textbooks on automata for details.

Given a language $L \subseteq X^*$, the reduced acceptor (unique up to isomorphisms) can be built using the equivalence classes of words in X^* with respect to the Nerode equivalence. The set of states is the set of equivalence classes $[w]_L$ with $w \in X^*$. This set is finite if and only if L is a regular language. The initial state is the class $[\varepsilon]_L$. The set F of accepting

states consists of exactly those classes $[w]_L$ for which $w \in L$. The transition function under input $x \in X$ leads from $[w]_L$ to $[wx]_L$. These definitions do not depend on the representatives of the equivalence classes. We refer to this construction as the *Nerode construction*[3]

We also need to consider equivalence of states of different acceptors and equivalence of different acceptors. This is most conveniently expressed for automata with outputs [Sta69], but can be re-phrased for acceptors as follows: Consider two acceptors $A = (Q, X, \delta, q_0, F)$ and $A' = (Q, X, \delta, q_0, F)$. States $q \in Q$ and $q' \in Q'$ are *equivalent*, $q \cong q'$, if and only if $L(A_q) = L(A'_{q'})$. The acceptor A is said to *simulate* A' if and only if, for every state $q' \in Q'$ there is a state $q \in Q$ with $q' \cong q$. The acceptors A and A' are said to be *universally equivalent*[4], $A \cong A'$, if and only if A and A' simulate each other.

Remark 3 *Let A and A' be universally equivalent acceptors, and let q and q' be equivalent states of A and A', respectively. Then, for all $i \in \mathbb{N}, L_i(A_q) = L_i(A'_{q'})$.*

We state an automaton theoretic characterisation of prefix codes, well-known in coding theory (see, for example, [BP85, JK97]), adapted to the present terminology. To make the presentation self-contained, we include a proof.

Proposition 1 *Let $A = (Q, X, \delta, q_0, F)$ be an acceptor with $q_0 \notin F$. Then $L_1(A) = \sqrt[\mathsf{Pref}]{L(A)}$ is a prefix code. Conversely, if $L \subseteq X^+$ is a non-empty prefix code then there is an acceptor A such that $L = L_1(A)$.*

Proof. If $|L_1(A)| \leq 1$, nothing needs to be proved. Assume, therefore, that $L_1(A)$ contains at least two distinct words u and w. If $L_1(A)$ is not a prefix code such words exist with u a proper prefix of w, that is, $w = uv$ for some $v \in X^+$. But then $w \in L_i(A)$ with $i \geq 2$, hence $w \notin L_1(A)$, a contradiction!

For the converse, consider the acceptor $A = (Q, X, \delta, q_0, F)$ defined as follows: Let $Q = (\mathsf{Pref}(L) \setminus L) \cup \{s\}$ with $q_0 = \varepsilon$, and s is a new symbol; let

[3] The Nerode construction works for any type of language, but need not be computable. It is computable if the Nerode equivalence is decidable

[4] Acceptors A and A' are equivalent in the usual sense if $L(A) = L(A')$. Universal equivalence is a far stronger requirement.

$F = \{\varepsilon\}$ and, for $q \in Q$ and $x \in X$, let

$$\delta(q,x) = \begin{cases} qx, & \text{if } qx \in \mathsf{Pref}(L) \setminus L, \\ \varepsilon, & \text{if } qx \in L, \\ s, & \text{if } q = s \text{ or } qx \notin \mathsf{Pref}(L), \text{ and} \\ \delta(\varepsilon,x), & \text{if } q \in L. \end{cases}$$

In essence, the transition function defines the tree of code words; however, the leaves are combined into the initial state ε which is also the only final state; the state s serves as a sink state. A word w is accepted if and only if $\delta(\varepsilon,w) = \varepsilon$. Thus $L(A) = L^*$.

As L is a prefix code, we have $L = L_1(A)$. ❑

Consider a property P of words in X^+ according to which words in X^* can be decomposed into factors. For example, if L is a prefix code, then every word in L^+ can be decomposed uniquely into a product of words in L, and every word in X^* can be decomposed into a product of words which are in L or which have no infix, which is in L. In this case, P would be defined as follows: A word $w \in X^+$ satisfies P if and only if $w \in L$. In general, a word may have any finite number of decompositions according to P. This idea is captured in the following definition.

Definition 4 Let P be a property of words in X^+. A P-decomposition of a word $w \in X^*$ is a construct $w = (n,\vec{u},\vec{v})$ with the following properties:

1. $n \in \mathbb{N}_0$.
2. $\vec{u} = (u_1,u_2,...,u_n)$ with $u_1,u_2,...,u_n \in X+$, each satisfying P.
3. $\vec{v} = (v_0,v_1,...,v_n)$ with $v_0,v_1,...,v_n \in X^*$ such that no v_i has an infix satisfying P.
4. $w = v_0u_1v_1u_2v_2\cdots u_nv_n$.

Such P-decompositions are required, for instance, for the decoding of encoded messages received over a noisy channel. The decoder will attempt to determine a decomposition of the received messages into words possibly related to code words, and then attempt to invert the encoding, possibly correcting errors. The first part of this process is modelled by an acceptor as follows.

Definition 5 Let A be an acceptor, let P be a property of words in X^+, let $w \in X^+$, and let $\vec{w} = (n,\vec{u},\vec{v})$ be a P-decomposition of w. Let

$$\mathsf{Pref}_P(\vec{w}) = \{v_0u_1v_1u_2v_2\cdots u_i | i = 1,2,\ldots,n\}.$$

The acceptor A *exposes* the P-decomposition $\vec{w}$ if and only of the following conditions are satisfied

1. $t \in L(A)$ for all $t \in \mathsf{Pref}_P(\vec{w})$.
2. $t \notin L(A)$ for all $t \in \mathsf{Pref}(w) \setminus \mathsf{Pref}_P(\vec{w})$.

Let A, P, w and $\vec{w}$ be as in Definition 5. If $n = 0$, then $\vec{u}$ is the empty sequence and $\mathsf{Pref}_P(\vec{w}) = \emptyset$. If $n > 0$, then the word $v_0u_1v_1u_2v_2\cdots u_i$ is accepted at stage i. The ends, but not the beginnings, of the infixes listed in $\vec{u}$ are recognised in the given order.

3 Restricted Infix Codes

In this section we provide new characterisations of p-infix and p-suffix codes. These characterisations may turn out to be interesting beyond the scope of the present paper. They resemble the fact that $L, \varepsilon \notin L$, is a prefix (suffix) code if and only if $L = \sqrt[\mathsf{Pref}]{LX^*}$ $\left(L = \sqrt[\mathsf{Suff}]{X^*L}\right)$.

Theorem 1 *Let $L \subseteq X^+$ and $\varepsilon \notin L$.*

1. *L is a p-infix code if and only if $L \subseteq \sqrt[\mathsf{Pref}]{X^*L}$.*
2. *L is a s-infix code if and only if $L \subseteq \sqrt[\mathsf{Suff}]{LX^*}$.*

Proof. We only prove the first statement. The second one follows by duality.

Suppose that L is a p-infix code, that is, $L \cap X^*LX^+ = \emptyset$. Then $L \subseteq X^*L \setminus X^*LX^+ = \sqrt[\mathsf{Pref}]{X^*L}$.

For the converse, assume that L is not a p-infix code. Let $u \in L \cap X^*LX^+$. Then there are words $w \in L$ and $v \in X^*$ such that $vw <_p u$. Then u is in L, but not in $\sqrt[\mathsf{Pref}]{X^*L}$. ❑

4 Solid Codes

Solid codes were introduced in [Lev64a] as *strongly regular codes*; they are called *codes without overlap* in [Lev70]. The term *solid codes* seems to appear in [SY90] for the first time. A combinatorial characterisation of solid codes is provided in [JY90]. Many properties of solid codes are summarised in [JK97].

There are two definitions of solid codes, reflecting two different ways of interpreting the same situation: one purely combinatorial; a second one implying error-resistance properties.

The first and earlier definition is purely combinatorial; it defines "strongly regular codes" in the sense of [Lev64a] or "codes without overlap" in the sense of [Lev70].

Definition 6 A *solid code* over X is a language $L \subseteq X^+$ which is an overlap-free infix code.

For the second definition we need the following auxiliary notion. For a language $L \subseteq X^+$ and a word $w \in X^*$, an L-decomposition of w is a construct $\vec{w} = (n, \vec{u}, \vec{v})$ with $n \in \mathbb{N}_0, \vec{u} = (u_1, u_2, ..., u_n)$ and $v = (v_0, v_1, \ldots, v_n)$ of words in X^*, such that

$$v_0 u_1 v_1 u_2 v_2 \cdots u_n v_n = w,$$

$u_1, u_2, \ldots, u_n \in L$ and, for $i = 0, 1, \ldots, n$, $\mathrm{Inf}(v_i) \cap L = \emptyset$. An L-decomposition is a special kind of P-decomposition according to Definition 4. For every $L \subseteq X^+$, every word in X^* has at least one L-decomposition.

Definition 7 A *solid code* over X is a language $L \subseteq X^+$ such that every word in X^* has a unique L-decomposition.

Theorem 2 ([JY90]) *Solid codes as defined in Definitions 6 and 7 are the same.*

There is a subtle difference between the two definitions of solid codes which becomes apparent when one attempts to relatives the concept in the following sense: The properties are no longer required to hold for all words in X^*, but only for words in a given language $M \subseteq X^*$. Intuitively, for Definition 6, L would behave as an overlap-free infix code only for words in M; similarly, for Definition 7, only the words in M would need to have unique L-decompositions. These ideas are explored in [DJKM12, Hea00, Jür09, Jür11].

By the standard relativisation technique proposed in [DJKM12], the relativised versions of the two definitions are not equivalent in general.

Within the hierarchy of classes of codes, the solid codes form a proper subclass of the comma-free codes. They are incomparable to the hypercodes. Solid hypercodes have superb error-detection and synchronisation capabilities.

For a constructively given linear language L, it is undecidable whether it is a solid code (see [JK97], Table 9.1 and Theorem 9.5). On the other hand, if L is regular (constructively given), then it is decidable whether L is a solid code (see [JY90] and [JK97], Table 9.1).

These results are extended and refined in [HS11]: For regular languages given by finite non-deterministic acceptors, polynomial time bounds, in terms of the number of states and the number of transitions, are derived for deciding the properties of being overlap-free or solid. For linear languages the property of being overlap-free is undecidable in general. From [JK97], Theorem 9.5, it is known that the property of being an infix code is undecidable for linear languages.

For further general information regarding various aspects of solid codes see [JY90, JK97, SY90, Yu05]. Specific properties of solid codes are discussed in the following publications: information rate of solid codes [JKL04, JK05, Lev70, Lev04]; maximality of solid codes [JKK01, Lâm01, Lâm03, KY10, JK06]; relativised solid codes [Hea00, DJKM12, Jür09, Jür11]; involution solid codes in the context of DNA computing [JKM06, KM06, JKM08]; information transmission [Bal02]; application to pattern matching [HW06][5].

5 State-Invariant Decoders for Finite Solid Codes

For finite solid codes a characterisation in terms of transducers is given in [Rom66]. This work is presented in more general terms in Chapter 11 of [JK97]. Here we provide a brief summary of these results.

Definition 8 A *finite deterministic transducer* is a construct $A = (Q, X, Y, d, m)$ with the following properties and interpretation:

1. Q is a finite non-empty set of states;
2. X is the input alphabet and Y is the output alphabet; X and Y are finite and non-empty;

[5] In the paper [HW06] it seems to be assumed that the languages under consideration are infix codes. Otherwise some of the statements about overlap-free languages would be incorrect. Moreover, in that paper a language is defined to be overlap-free if and only no two distinct words overlap (Definition 1), thus allowing for the presence of words which overlap themselves. The presence of such words is probably not intended as it would invalidate several results of that paper. This is readily corrected if one uses the standard definition of overlap-free languages, hence solid codes.

3. $d : Q \times X \to Q$ is the transition function;

4. $m : Q \times X \to Y^*$ is the output function.

The behaviour of a transducer on input words is defined by sequentiality as usual:

$$d(q,w) = \begin{cases} q, & \text{if } w = \varepsilon, \\ d(d(q,x),w_0), & \text{if } w = xw_0 \text{ with } x \in X \end{cases}$$

and

$$m(q,w) = \begin{cases} \varepsilon, & \text{if } w = \varepsilon, \\ m(q,x)m(d(q,x),w_0), & \text{if } w = xw_0 \text{ with } x \in X. \end{cases}$$

Definition 9 Let X and Y be alphabets and $L \subseteq Y^+$ with $|L| = |X|$. Let f be a bijection of X onto L. A *state-invariant decoder for f without look-ahead* is a finite deterministic transducer $A = (Q,X,Y,d,m)$ with the following properties:

1. For all $q \in Q$ and all $v \in L, m(q,v) = f^{-1}(v)$.

2. For all $q \in Q$, all $v \in L$ and all $u \in \mathsf{Pref}_+(v), m(q,u) = \varepsilon$.

A state-invariant transducer without look-ahead for f produces exactly the decodings of the words in L; these are produced regardless of the initial state of the transducer and precisely at the time when the last symbol of a word in L has been read. If such a transducer reads an arbitrary word $w \in Y^+$, the output is the concatenation of the decodings of those words in L, which it encounters as disjoint infixes of w. This establishes an intuitive connection with Definition 7.

Theorem 3 ([Rom66]) *Let X and Y be alphabets and $L \subseteq Y^+$ with $|L| = |X|$. Let f be a bijection of X onto L. Then L is solid code if and only if there is a state-invariant decoder without look-ahead for f.*

A detailed proof of this theorem is given in [JK97]. The main construction, due to Levenshtein [Lev64a] and Romanov [Rom66] is described in the next sections of this paper. There we also point out connections to related ideas in combinatorics on words or stringology.

6 Levenshtein-Romanov Mapping

In our construction of acceptors for solid codes, the following mapping, due to Levenshtein [Lev64a] and Romanov [Rom66], is essential.

Definition 10 Let X be an alphabet and let $L \subseteq X^+$ with $L \neq \emptyset$. The mapping $\sigma_L : X^* \to X^*$ defined by

$$\sigma_L(w) = \max_{\leq_s}(\mathsf{Suff}(w) \cap \mathsf{Pref}(L))$$

is called the *Levenshtein-Romanov mapping* for L.

When σ_L is used in the sequel we assume, without special mention, that L is non-empty and that L does not contain the empty word.

Proposition 2 *Let $L \subseteq X^+$ with $L \neq \emptyset$ and let $w, w' \in X^*$. The Levenshtein-Romanov mapping σ_L has the following properties:*

1. $\sigma_L(w) \leq_s w$, *and if* $w' \in \mathsf{Pref}(L)$ *and* $w' \leq_s w$ *then* $w' \leq_s \sigma_L(w)$.
2. *If* $w' \leq_s w$ *then* $\sigma_L(w') \leq_s \sigma_L(w)$*; in particular,* $\sigma_L(\varepsilon) = \varepsilon$.
3. $w \in \mathsf{Pref}(L)$ *if and only if* $\sigma_L(w) = w$.
4. $\sigma_L(w) = \sigma_L(\sigma_L(w))$.
5. *If* $\sigma_L(w) \leq_s w' \leq_s w$ *then* $\sigma_L(w) = \sigma_L(w')$.
6. $\sigma_L(ww') = \sigma_L(\sigma_L(w)w')$.
7. $\sigma_L(w) \in X^*L$ *if and only if* $w \in X^*L$.

Proof.

1. This follows from $w', \sigma_L(w) \in \mathsf{Suff}(w) \cap \mathsf{Pref}(L)$.
2. Also obvious by the definition of σ_L.
3. If $w \in \mathsf{Pref}(L)$ then $w \in \mathsf{Suff}(w) \cap \mathsf{Pref}(L)$, hence $\sigma_L(w) = w$. Conversely, as $w = \max_{\leq_s}(\mathsf{Suff}(w))$, $\sigma_L(w) = w$ implies $w \in \mathsf{Pref}(L)$.
4. As $\sigma_L(w) \in \mathsf{Pref}(L)$, $\sigma_L(w) = \sigma_L(\sigma_L(w))$ by (3).
5. We have
$$\sigma_L(\sigma_L(w)) \leq_s \sigma_L(w') \leq_s \sigma_L(w)$$
by (2). By (4), $\sigma_L(\sigma_L(w)) = \sigma_L(w)$. Thus $\sigma_L(w) = \sigma_L(w')$.

6. First we prove that $\sigma_L(ww') \leq_s \sigma_L(w)w'$. Assume the contrary. As $\sigma_L(ww') \leq_s ww'$ and $\sigma_L(w)w' \leq_s ww'$, there is a word $u \in X^+$ such that $\sigma_L(ww') = u\sigma_L(w)w' \leq_s ww'$. As $\sigma_L(ww') \in \mathsf{Pref}(L)$, also $u\sigma_L(w) \in \mathsf{Pref}(L)$. Moreover, $u\sigma_L(w) \leq_s w$. As $\sigma_L(w)$ is the longest suffix of w which is a prefix of L, one has $\sigma_L(w) \leq_s u\sigma_L(w)$, but $u \neq \varepsilon$, a contradiction! As a consequence one has

$$\sigma_L(ww') \leq_s \sigma_L(w)w' \leq_s ww'.$$

Using (5) this proves that $\sigma_L(ww') = \sigma_L(\sigma_L(w)w')$.

7. Let $w = uv$ where $v \in L$. Then $v = \sigma_L(v) \leq_s \sigma_L(w)$, that is $\sigma_L(w) \in X^*L$. The other direction follows from $\sigma_L(w) \leq_s w$.

❑

The statements of Proposition 2 can be interpreted as follows: The Levenshtein-Romanov mapping σ_L maps X^* onto $\mathsf{Pref}(L)$; it is the identity exactly on $\mathsf{Pref}(L)$. With respect to $\leq_s$, the mapping is monotone. By the sixth statement, it is sequential. The fifth property can be understood to express some kind of continuity.

The following statement is a direct consequence of the definition of σ_L.

Lemma 1 *Let $L \subseteq X^+, L \neq \emptyset$. Then $\sigma_L(w) \in \mathsf{Pref}(L) \cap X^*L$ for all $w \in X^*L$.*

In connection with Proposition 2 the question arises whether $\sigma_L(w) \in L$ for all $w \in X^*L$. We answer this in the following.

Proposition 3 *Let $L \subseteq X^+$ with $L \neq \emptyset$. The following conditions are equivalent.*

1. $\mathsf{Pref}(L) \cap X^*L = L$.
2. $\sigma_L(w) \in L$ *for all* $w \in X^*L$.

Proof. Let $v \in (\mathsf{Pref}(L) \cap X^*L) \setminus L$. Since $v \in \mathsf{Pref}(L)$, we have $\sigma_L(v) = v \notin L$.

Assume $\sigma_L(w) \notin L$ for some $w \in X^*L$. Then $\sigma_L(w) \in X^*L$ by Lemma 1. By definition $\sigma_L(w) \in \mathsf{Pref}(L)$. Thus $\sigma_L(w) \in (\mathsf{Pref}(L) \cap X^*L) \setminus L$.

❑

The condition of $\mathsf{Pref}(L) \cap X^*L = L$ in Proposition 3 leads to new insights into the structure of certain kinds of prefix or suffix codes as discussed in the sequel.

The Levenshtein-Romanov mapping was introduced in 1964; it is essential for the construction of state-invariant decoders without look-ahead for finite solid codes. It was re-introduced 10 years later by Aho and Corasick as the failure function in their algorithm for string matching [AC75]. Various versions of this algorithm are presented and analysed in [BPR10, CH97]. In this part of the literature the resulting automaton for a finite set of "patterns" is known as the Aho-Corasick automaton, the string-matching automaton, the dictionary-matching automaton, the dictionary automaton or the pattern-matching machine. Related constructs are used in [CH85, Hof84] to formulate algorithms for deciding the unique decipherability of a given finite language used as a code. Aho-Corasick automata have the same underlying transition structure as the state-invariant transducers without look-ahead for finite solid codes. The underlying ideas of Aho-Corasick automata are also found in Levenshtein's 1964 paper on properties of coding and self-adjusting automata [Lev64b]. Some parts of Proposition 2 are proved implicitly in [Lev64a, Rom66, JK97] or explicitly as Lemma 5.1 of [CH97].

7 The Condition $\mathsf{Pref}(L) \cap X^*L = L$ and Its Dual

The conditions that $\mathsf{Pref}(L) \cap X^*L = L$ or its dual $\mathsf{Suff}(L) \cap LX^* = L$ have unexpected consequences, some of which are explored in the present section.

Lemma 2 *Let $L \subseteq X^+, L \neq \emptyset$. The following statements hold true:*

1. *If $X^+LX^+ \cap L = \emptyset$ then $\mathsf{Pref}(L) \cap X^*L = L$.*
2. *If L is a prefix code then $\mathsf{Pref}(L) \cap X^*L = L$ implies $X^*LX^+ \cap L = \emptyset$.*
3. *If L is a suffix code then $\mathsf{Pref}(L) \cap X^*L = L$ implies $X^+LX^* \cap L = \emptyset$.*

Proof. (1) Clearly $L \subseteq \mathsf{Pref}(L) \cap X^*L$. Consider $u \in \mathsf{Pref}(L) \cap X^*L$ with $u \notin L$. Then there is $w \in X^+$ such that $uw \in L$. As $u \in X^*L \setminus L$, one has $u = v'v$ with $v \in L$ and $v' \in X^+$. Thus $uw = v'vw \in X^+LX^+ \cap L$, a contradiction.

(2) Let L be a prefix code and consider $u \in X^*LX^+ \cap L$. Then $u = v'vw$ with $v' \in X^*, v \in L$ and $w \in X^+$. If $v' = \varepsilon$, then $u = vw$ with $u, v \in L$, and $w \in X^+$, contradicting the fact that L is a prefix code. Hence $v' \neq \varepsilon$ and $v'v \notin L$ as $u \in L$. Therefore, $v'v \in (\mathsf{Pref}(L) \cap X^*L) \setminus L$, a contradiction.

(3) Let L be a suffix code and consider $u \in X^+LX^* \cap L$. Then $u = v'vw$ with $v' \in X^+, v \in L$, and $w \in X^*$. As L is a suffix code $v'v \notin L$. Therefore, $v'v \in (\mathsf{Pref}(L) \cap X^*L) \setminus L$. ❑

Each of the conditions $X^*LX^+ \cap L = \emptyset$ and $X^+LX^* \cap L = \emptyset$ implies that L is uniquely decodable, that is, a code. In contrast, the condition $X^+LX^+ \cap L = \emptyset$ does not imply this: The language $L = \{ab, abab\}$ over $X = \{a,b\}$ satisfies the condition, but is not a code.

The converse of Lemma 2 (1) is not true in general, not even for codes. Consider $L = \{ab, abc, b\}$ with $X = \{a,b,c\}$. The language L is a code. One has $\mathsf{Pref}(L) = \{a, ab, abc, b, \varepsilon\}$. Hence $\mathsf{Pref}(L) \cap X^*L = \{ab, abc, b\} = L$. But $abc \in X^+LX^+ \cap L$. This same example shows also that Statements (2) and (3) of the same lemma do not hold true for arbitrary codes. Corollary 1 below shows, that the implications are actually equivalences.

By left-right duality one obtains the following results.

Lemma 3 *Let $L \in X^*$. The following statements hold true:*

1. *If $X^+LX^+ \cap L = \emptyset$ then $\mathsf{Suff}(L) \cap LX^* = L$.*
2. *If L is a suffix code then $\mathsf{Suff}(L) \cap LX^* = L$ implies $X^+LX^* \cap L = \emptyset$.*
3. *If L is a prefix code then $\mathsf{Suff}(L) \cap LX^* = L$ implies $X^*LX^+ \cap L = \emptyset$.*

Corollary 1 *Let $L \subseteq X^+$ with $L \neq \emptyset$.*

1. *L is a p-infix code if and only if L is a prefix code and $\mathsf{Pref}(L) \cap X^*L = L$.*
2. *L is a p-infix code if and only if L is a prefix code and $\mathsf{Suff}(L) \cap LX^* = L$.*
3. *L is an s-infix code if and only if L is a suffix code and $\mathsf{Pref}(L) \cap X^*L = L$.*
4. *L is an s-infix code if and only if L is a suffix code and $\mathsf{Suff}(L) \cap LX^* = L$.*

Proof. We prove only the first statement. The remaining ones follow by duality.

Let L be a p-infix code, that is, $L \cap X^*LX^+ = \emptyset$. Then $L \cap X^+LX^+ = \emptyset$, hence $\mathsf{Pref}(L) \cap X^*L = L$ by Lemma 2 (1).

Conversely, let L be a prefix code and $\mathsf{Pref}(L) \cap X^*L = L$. Then $X^*LX^+ \cap L = \emptyset$, that is, L is a p-infix code. ❑

These results lead to new characterisations of s-infix codes and overlap-free prefix codes as follows.

Theorem 4 *Let $L \subseteq X^+$ with $L \neq \emptyset$. L is an s-infix code if and only if $\sigma_L(uv) = v$ for all $u \in X^*$ and $v \in L$.*

Proof. Let L be an s-infix code. Then L is a suffix code and $\mathsf{Pref}(L) \cap X^*L = L$ by Corollary 1 (3). Let $u \in X^*$ and $v \in L$. Then $uv \in X^*L$, hence $\sigma_L(uv) \in L$ by Proposition 3. As $v \in L$ one has $v \leq_s \sigma_L(uv)$. Thus $v = \sigma_L(uv)$ as L is a suffix code.

For the converse, assume that $\sigma_L(uv) = v$ for all $u \in X^*$ and $v \in L$. Suppose that $uv \in L$. Then $\sigma_L(uv) = uv$ by Proposition 2. Hence $u = \varepsilon$, that is, L is a suffix code. To prove that L is an s-infix code, we need to show that $\mathsf{Pref}(L) \cap X^*L = L$. The inclusion $L \subseteq \mathsf{Pref}(L) \cap X^*L$ is true by definition. For the converse inclusion consider $w \in \mathsf{Pref}(L) \cap X^*L$. Then $w = uv$ for some $u \in X^*$ and $v \in L$. Hence $\sigma_L(w) = \sigma_L(uv) = v$. On the other hand, by the definition of $\sigma_L, uv = w \in \mathsf{Pref}(L)$ implies $\sigma_L(w) = w = uv$. Thus $w = uv = v \in L$. ❑

Theorem 5 *Let $L \subseteq X^+$ with $L \neq \emptyset$. L is an overlap-free prefix code if and only if $\sigma_L(uv) = \sigma_L(v)$ for all $u \in L$ and $v \in X^+$.*

Proof. Let L be an overlap-free prefix code. Consider $u \in L$ and $v \in X^+$. Thus $\sigma_L(v) \leq_s v <_s uv$. By Proposition 2 one has $\sigma_L(v) \leq_s \sigma_L(uv) \leq_s uv$.

We now prove that $\sigma_L(uv) \leq_s v$. Assume the contrary, that is, $v <_s \sigma_L(uv)$. As L is a prefix code, it follows from $u \in L, v \in X^+$ and $\sigma_L(uv) \in \mathsf{Pref}(L)$ that $\sigma_L(uv) \neq uv$. Thus $uv = u'\sigma_L(uv)$ for some $u' \in X^+$ with $u' <_p u$. Hence there is a word $w \in L$ with $\sigma_L(uv) \leq_p w$ such that u and w overlap, contradicting the assumption that L is overlap-free.

Thus $\sigma_L(v) \leq_s \sigma_L(uv) \leq_s v$. By Proposition 2 this implies $\sigma_L(v) = \sigma_L(uv)$.

Conversely, assume that $\sigma_L(uv) = \sigma_L(v)$ for all $u \in L$ and all $v \in X^+$. Suppose that L not a prefix code or not overlap-free.

If L is not a prefix code, then there are $u \in L$ and $v \in X^+$ such that $uv \in L$. Thus $\sigma_L(v) = \sigma_L(uv) = uv$, hence $uv = u$, a contradiction.

If L is not overlap-free, there are non-empty words u, v, w such that $uv \in L$ and $vw \in L$. Thus $w <_s vw <_s uvw$. Hence $\sigma_L(uvw) = \sigma_L(w) \leq_s w$ and $vw \leq_s \sigma_L(vw) = vw$, again a contradiction! ❑

In the sequel we use a combination of the premises of Theorems 4 and 5. We consider languages, which are both s-infix codes and overlap-free

prefix codes. Such languages are precisely the solid codes.

8 Fault-Tolerant Acceptors for Solid Codes

We now consider fault-tolerant acceptors for solid codes. The underlying idea is derived from Definition 9, Theorem 3 and the proof of that theorem. Those constructions rely on the assumption that the solid codes under consideration are finite, allowing one to consider a specific natural encoding. We drop this assumption. Consequently, without any knowledge about the encoding, we cannot expect to obtain a decoder, but only an acceptor. In general, this acceptor can be infinite. For finite solid codes the acceptor is similar to a finite state-invariant transducer without look-ahead. For arbitrary solid codes we expect to obtain an acceptor with special properties which correspond to state-invariance and the lack of look-ahead. Moreover, we expect these properties to be preserved when the acceptor is reduced. This would guarantee that a solid code, which is regular (or rational) as a language, is accepted by a finite acceptor with these special properties.

We follow the structure of Proposition 1, the characterisation of prefix codes by acceptors: In Theorem 6 we state that every acceptor A defines a solid code: this code is the suffix root of the language consisting of all those words which are accepted by A regardless of the initial state. In Theorem 8 we state that every solid code L defines an acceptor A such that the solid code L' defined by A coincides with L.

Theorem 6 *Let $A = (Q, X, \delta, q_0, F)$ be a deterministic acceptor. The following statements hold true:*

1. *The language*
$$\bigcap_{q \in Q} L_1(A_q)$$
is an overlap-free p-infix code.

2. *The language*
$$\sqrt[\mathrm{Suff}]{\bigcap_{q \in Q} L_1(A_q)}$$
is a solid code.

Proof. Let
$$L' = \bigcap_{q \in Q} L_1(A_q) \text{ and } L = \sqrt[\mathrm{Suff}]{L'}$$
We need to prove that L' is a p-infix code and overlap-free.

By Proposition 1, the languages $L_1(A_q)$ are prefix codes for all $q \in Q$. Hence, also their intersection L' is a prefix code.

Suppose that L' is not a p-infix code. Then there are words $v, w \in L'$ and $u_1, u_2 \in X^+$ such that $w = u_1 v u_2$. As $v, w \in L'$, one has $\delta(q, v) \in F$ and $\delta(q, w) \in F$ for all $q \in Q$. Therefore, also $\delta(q, u_1 v) = \delta(\delta(q, u_1), v) \in F$. The fact that $u_1 v <_p w$ implies that $w \notin L_1(A_q)$, a contradiction!

Now suppose that L' is not overlap-free. Then there are words $v, w \in L'$, not necessarily distinct, which overlap. Without loss of generality, we assume there are words $u_1, u_2, u \in X^+$ such that $w = u_1 u$ and $v = u u_2$. As $v, w \in L'$, one has

$$\delta(q, w) = \delta(\delta(q, u_1), u) \in F \text{ and } \delta(\delta(q, u_1), v) \in F$$

for all $q \in Q$. The fact that $u <_p v$ implies that $v \notin L_1(A_{\delta(q,u_1)})$, a contradiction!

This proves, that L' is an overlap-free p-infix code. As $L \subseteq L'$, also L is an overlap-free p-infix code. By definition, L is also a suffix code. Hence, L is a solid code. ❑

In Theorem 6 we include the situation when $L_1(A_q) = \emptyset = L(A_q)$. This is not very useful, but also means that such states q should be discarded.

Definition 11 Let $A = (Q, X, \delta, q_0, F)$ be a deterministic acceptor. Define

$$L^{\mathrm{ol-pi}}(A) = \bigcap\nolimits_{q \in Q} L_1(A_q)$$

and

$$L^{\mathrm{solid}}(A) = \sqrt[\mathrm{Suff}]{L^{\mathrm{ol-pi}}(A)}$$

Theorem 7 *Let A and A' be universally equivalent acceptors. Then $L^{\mathrm{ol-pi}}(A) = L^{\mathrm{ol-pi}}(A')$ and $L^{\mathrm{solid}}(A) = L^{\mathrm{solid}}(A)$.*

Proof. The first statement follows from Remark 3. The second statement is an immediate consequence. ❑

In the statements of Theorem 6 the intersection expresses a kind of state invariance; taking the suffix root can be interpreted as avoiding look-ahead.

The following example shows that taking the suffix root is essential.

Example 1 Let $A = (\{e, f\}, \{a, b\}, \delta, e, \{f\})$ where $\delta(q, a) = e$ and $\delta(q, b) = f$ for $q \in \{e, f\}$. Then $\bigcap_{q \in \{e,f\}} L_1(A_q) = a^* b$ which is an overlap-free p-infix

code but not a suffix code, hence not a solid code. The suffix root of the intersection is the singleton language $\{b\}$, a trivial solid code over the alphabet $\{a,b\}$. The corresponding automaton is shown in Fig. 2. ❑

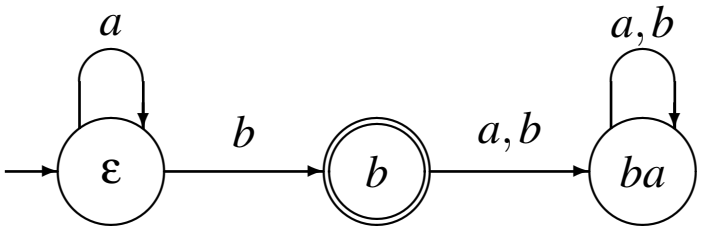
By Theorem 6, every deterministic acceptor, finite or infinite, describes a solid code as the suffix root of the language accepted regardless of the initial state. We now show that every solid code, regardless of its cardinality, is defined by an acceptor in this fashion. To our knowledge, this construction was first proposed in [Lev64a, Rom66], but only for finite solid codes; for those the acceptor can be converted into a state-invariant decoder without look-ahead in the sense of Theorem 3. For arbitrary finite languages the resulting finite acceptor is essentially the Aho-Corasick automaton (or string-matching automaton, dictionary automaton or pattern matching machine, etc.). In the sequel, in considering a language L over X, we only assume that $L \subseteq X^+$ and $L \neq \emptyset$. We do not assume that L is finite or has any simple computational structure; not even that L is recursively enumerable. Certainly, to perform concrete constructions one would have to make additional assumptions about L.

Definition 12 Let $L \subseteq X^+$ with $L \neq \emptyset$. The Levenshtein-Romanov acceptor for L is the acceptor $A = (Q, X, \delta, q_0, F)$ defined as follows:

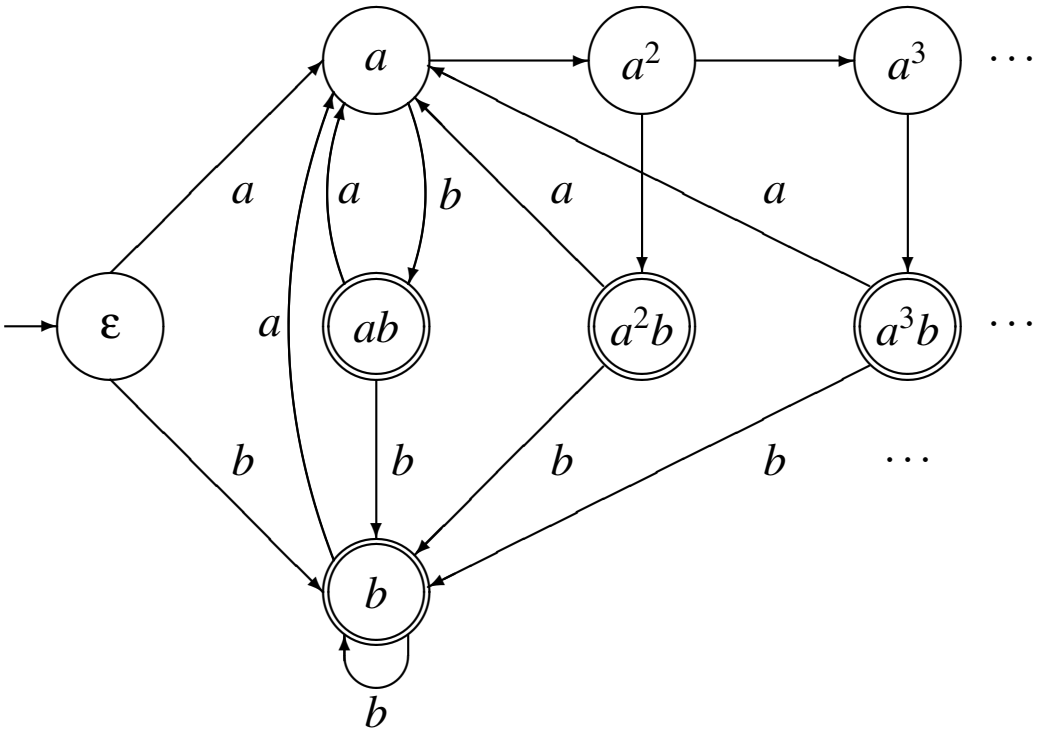
1. $Q = \mathsf{Pref}(L)$;
2. $q_0 = \varepsilon$;
3. $F = L$;
4. $\delta(q,x) = \sigma_L(qx)$ for $q \in Q$ and $x \in X$.

The language $L(A)$ accepted by the Levenshtein-Romanov acceptor A for a language L is usually not equal to L. The difference will be explained further below. The reduced acceptor obtained from A is called the reduced Levenshtein-Romanov acceptor for L.

Example 2 Let $X = \{a,b\}$ and $L = a^*b$. The reduced acceptor for L has three states. The Levenshtein-Romanov acceptor for L has infinitely many states. These acceptors are shown in Fig. 3. The languages of these acceptors are L and X^*L, respectively. The discussion below clarifies, why these languages are different. The reduced Levenshtein-Romanov acceptor for L is shown in Fig. 4. ❑



Reduced acceptor for a^*b



Levenshtein-Romanov acceptor for a^*b

Figure 3: The reduced acceptor and the Levenshtein-Romanov acceptor for the language a^*b of Example 2; heavy circles indicate accepting states.

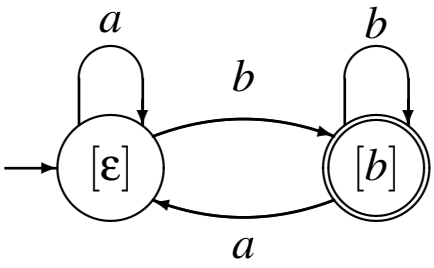


Figure 4: Reduced Levenshtein-Romanov acceptor for the language a^*b of Example 2.

The language accepted by the Levenshtein-Romanov acceptor for L with $\emptyset \neq L \subseteq X^+$ turns out to be a subset of X^*L. The details are as follows:

Proposition 4 *Let $L \subseteq X^+$ with $L \neq \emptyset$. Let $A = (Q, X, \delta, q_0, F)$ be the Levenshtein-Romanov acceptor for L.*

1. *For all $q \in Q$ and all $w \in X^*$, one has $\delta(q,w) = \sigma_L(qw)$.*
2. *$L \subseteq L(A) \subseteq X^*L$.*
3. *$\mathrm{Pref}(L) \cap X^*L = L$ if and only if $L(A) = X^*L$.*
4. *If $X^+LX^+ \cap L = \emptyset$ then $L(A) = X^*L$.*

Proof.

1. As $q \in \mathrm{Pref}(L)$, one has $\sigma_L(q) = q$ by Proposition 2. For $w = \varepsilon$, one has
$$\delta(q,w) = q = \sigma_L(q) = \sigma_L(qw).$$
Consider $w = vx$ with $v \in X^+$ and $x \in X$ and assume that $\delta(q,v) = \sigma_L(qv)$. Then
$$\delta(q,w) = \delta(q,vx) = \delta(\delta(q,v),x) = \sigma_L(\sigma_L(qv)x) = \sigma_L(qvx) = \sigma_L(qw)$$
by Proposition 2.

2. For $w \in L$ one has $\sigma_L(w) = w$, hence $w \in L(A)$. If $w \in L(A)$ then $\delta(\varepsilon,w) = \sigma_L(w) \in F = L$, hence $w \in X^*L$, again by Proposition 2.

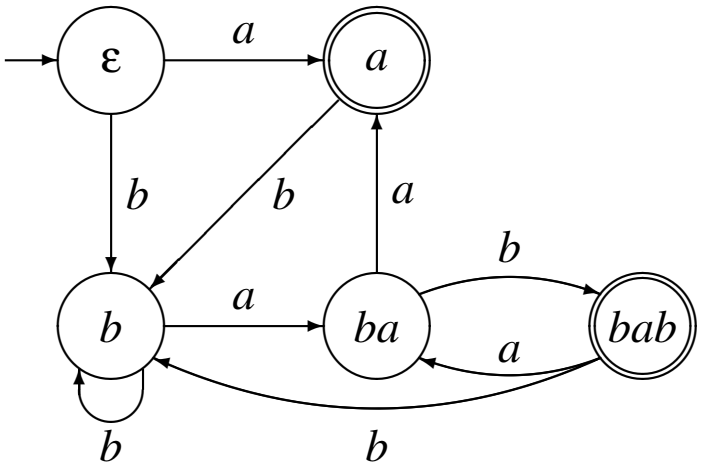


Figure 5: The Levenshtein-Romanov acceptor for the language L of Example 3.

3. One has $w \in L(A)$ if and only if $\sigma_L(w) \in L$ by the definition of A. If $L(A) = X^*L$ then $w \in X^*L$ implies $\sigma_L(w) \in L$, hence $\mathsf{Pref}(L) \cap X^*L = L$ by Proposition 3. Conversely, if $\mathsf{Pref}(L) \cap X^*L = L$, then $\sigma_L(w) \in L$ for all $w \in X^*L$ by Proposition 3. This implies $w \in L(A)$.

4. This follows from Lemma 2.1 and 3. ❑

Example 3 Consider the language $L = \{a, bab\}$ over the alphabet $X = \{a, b\}$. The reduced Levenshtein-Romanov acceptor A for L is shown in Fig. 5. One has $\mathsf{Pref}(L) \cap X^*L = \{a, ba, bab\}$. As this differs from L, one has $L(A) \neq X^*L$. For example, $ba \in X^*L \setminus L(A)$.

Consider $L' = \{a, ba, bab\}$ instead. The Levenshtein-Romanov acceptor A' for L' is obtained from A by making also the state ba accepting. Now $\mathsf{Pref}(L') \cap X^*L' = L'$ and, hence $L(A') = X^*L'$. However, $X^+L'X^+ \cap L' = \{bab\}$. This shows that the converse of Proposition 4 (4) is not true in general. ❑

Lemma 4 *Let $L \subseteq X^+$ with $L \neq \emptyset$, and let $A = (\mathsf{Pref}(L), \varepsilon, \delta, L)$ be its Levenshtein-Romanov acceptor. Then the following statements hold true:*

1. *A is initially connected and has no useless states.*

2. *A is strongly connected, if and only if for every word $u \in L$ there is a word $v \in X^+$ such that $\mathsf{Suff}(uv) \cap \mathsf{Pref}(L) = \{\varepsilon\}$.*

3. *If* $\mathsf{Pref}(L) \supseteq X$ *then* A *is not strongly connected.*

4. *If* L *is finite and* $\mathsf{Pref}(L) \not\supseteq X$ *then* A *is strongly connected.*

5. *If* L *is an overlap-free prefix code with* $X \not\subseteq \mathsf{Pref}(L)$*, then* A *is strongly connected.*

Proof.

1. The statement is a direct consequence of Definition 12.

2. By Proposition 4 (1), one has $\delta(u,v) = \sigma_L(uv)$. Hence, $\sigma_L(uv) = \varepsilon$ if $\mathsf{Suff}(uv) \cap \mathsf{Pref}(L) = \{\varepsilon\}$. By (1) the acceptor A is initially connected. Therefore, it is strongly connected. Otherwise, one cannot return to the state ε.

3. Since $\mathsf{Pref}(L) \supseteq X$, we have $\mathsf{Suff}(uv) \cap \mathsf{Pref}(L) \cap X \neq \emptyset$ for every $v \in X^+$.

4. Let $y \in X \setminus \mathsf{Pref}(L)$ and let $m = \max\{|w| : w \in L\}$. Then $\mathsf{Pref}(L) \cap \mathsf{Suff}(X^* \cdot y^{m+1}) = \emptyset$. Thus $\sigma_L(u \cdot y^{m+1}) = \varepsilon$ for every $u \in X^*$.

5. Let $y \in X \setminus \mathsf{Pref}(L)$ and consider $u \in \mathsf{Pref}(L)$. Then there is a $v \in X^*$ such that $uv \in L$. Now Theorem 5 implies $\sigma_L(uvy) = \sigma_L(y) = \varepsilon$, that is, $\delta(u,vy) = \varepsilon$.

❑

From Lemma 3 one concludes that the Levenshtein-Romanov acceptor for a non-empty language $L \in \mathcal{L}_x \cap \mathcal{L}_{\mathrm{ol}}$ with $x \in \{\mathrm{p}, \mathrm{pi}, \mathrm{i}\}$ is strongly connected if and only if $\mathsf{Pref}(L) \not\supseteq X$. This holds, in particular, for a non-empty solid code L.

The language $L = a^*b$ over $X = \{a,b\}$ of Example 2 is an overlap-free prefix code containing b. Note that the word b has no proper prefixes or suffixes, hence does not overlap any word in the language including itself. As $\{a,b\} \subseteq \mathsf{Pref}(L)$ its Levenshtein-Romanov acceptor, shown in Fig. 3, is not strongly connected. The reduced acceptor shown in Fig. 4, however, is strongly connected.

Corollary 2 *Let* $L \subseteq X^+$ *with* $L \neq \emptyset$*, and let* A *be the Levenshtein-Romanov acceptor for* L*. For* $x \in \{\mathrm{pi}, \mathrm{si}, \mathrm{i}, \mathrm{solid}\}$*, if* $L \in \mathcal{L}_x$ *then* $L(A) = X^*L$*.*

Proof. In each case $X^+LX^+ \cap L = \emptyset$. ❑

Corollary 3 *Let* $L \subseteq X^+$ *with* $L \neq \emptyset$*, and let* A *be the Levenshtein-Romanov acceptor for* L*. If* $\mathsf{Pref}(L) \cap X^*L = L$ *and* L *is regular then also* $L(A)$ *is regular.*

Proof. The assumption implies that $L(A) = X^*L$. With L also X^*L is regular. ❑

The Levenshtein-Romanov acceptor A for a language L is finite if and only if $\mathsf{Pref}(L)$ is finite, hence, if and only if L is finite. Thus, when L is an infinite regular language satisfying the condition $\mathsf{Pref}(L) \cap X^*L = L$, the acceptor A is infinite, but $L(A) = X^*L$ is regular. As we want to characterise fault-tolerant acceptors for solid codes we have to investigate how the special properties of the Levenshtein-Romanov acceptor are translated into properties of the equivalent reduced acceptor.

Remark 4 *The condition* $\mathsf{Pref}(L) \cap X^*L = L$ *does not imply that* L *is a code.*

Consider the language $L = \{ab, abab\}$ *which is not a code. Then* $\mathsf{Pref}(L) = \{\varepsilon, a, ab, aba, abab\}$*, and, consequently,* $\mathsf{Pref}(L) \cap X^*L = L$*. The same language also satisfies* $X^+LX^+ \cap L = \emptyset$*.*

Proposition 5 *Let* $L \subseteq X^+$ *be a non-empty overlap-free prefix code. Let* A *be the Levenshtein-Romanov acceptor for* L*. Then,*

$$L_i(A_w) = L_i(A) \text{ and } L(A_w) \setminus \{\varepsilon\} = L(A)$$

for all $w \in L$ *and all* $i \in \mathbb{N}$*.*

Proof. Consider $w \in L$ and $i \in \mathbb{N}$. By Theorem 5, one has

$$\delta(\varepsilon, v) = \sigma_L(v) = \sigma_L(wv) = \delta(w, v)$$

for all $v \in X^+$. Thus,

$$v \in L_i(A) \text{ if and only if } v \in L_i(A_w)$$

and

$$v \in L(A) \text{ if and only if } v \in L(A_w).$$

This proves the statements taking into account that $\varepsilon \notin L(A)$, but $\varepsilon \in L(A_w)$. ❑

Thus the accepting states of the Levenshtein-Romanov acceptor A for an overlap-free prefix code essentially reset the acceptor to the initial state. When a long word is read, once an accepting state is reached a corresponding output can be generated, and the reading continues with the rest of the input starting again in the initial state ε.

Proposition 6 *Let* $L \subseteq X^+$ *with* $L \neq \emptyset$*, and let* A *be the Levenshtein-Romanov acceptor for* L*. The following statements hold true:*

1. *If L is an s-infix code, then $L \subseteq L(A_q)$ for all $q \in \mathsf{Pref}(L) = Q$.*
2. *If L is a solid code, then $L \subseteq L_1(A_q)$ for all $q \in \mathsf{Pref}(L) = Q$.*

Proof. Let $w \in L$.

1. For $q = \varepsilon$, one has $L(A_q) = L$, hence, trivially $w \in L(A_q)$. Let L be an s-infix code. Assume that $q \neq \varepsilon$. Then

$$\delta(q, w) = \sigma_L(qw) = w$$

   by Theorem 4 as L is an s-infix code. This proves that $w \in L(A_q)$.

2. Let L be a solid code. Then L is a prefix code. Thus, $L \subseteq L_1(A)$. We show that $\sigma_L(qv) \notin L$ for all $v \in \mathsf{Pref}_+(w)$. This would prove that $w \in L_1(A_q)$.

   Suppose otherwise. Then $w = vu$ for some $u, v \in X^+$. Consider $\delta(q, v) = \sigma_L(qv)$. By assumption $\sigma_L(qv) \in L$. There are two cases:

   (a) $\sigma_L(qv) \leq_s v$: Then $\sigma_L(qv)$ is an infix of $vu = w$, contradicting the fact that L is an infix code.

   (b) $v <_s \sigma_L(qv)$: Then $\sigma_L(qv)$ and $vu = w$ overlap, contradicting the fact that L is a solid code. ❑

By combining the statements of Propositions 5 and 6 one obtains the following property of the Levenshtein-Romanov construction.

Theorem 8 *Let $L \subseteq X^+$ be a non-empty solid code, and let A be the Levenshtein-Romanov acceptor for L. Then*

$$L = \sqrt[\mathsf{Suff}]{\bigcap_{q \in \mathsf{Pref}(L)} L_1(A_q)}.$$

Proof. One has $L(A) = X^*L$ by Corollary 2. By Proposition 5, $L \subseteq L_1(A_q)$ for all $q \in Q = \mathsf{Pref}(L)$. Hence

$$L \subseteq \sqrt[\mathsf{Suff}]{\bigcap_{q \in \mathsf{Pref}(L)} L_1(A_q)}.$$

The intersection is a subset of $L_1(A)$, hence of $L(A) = X^*L$. As L is also a suffix code, $\sqrt[\mathsf{Suff}]{L(A)} = L$. This proves the equality as claimed. ❑

From Propositions 6 and 8 we obtain a characterisation of solid codes by their fault-tolerant decoders.

Corollary 4 *A language $L \subseteq X^+$ is a non-empty solid code if and only if there is an acceptor $A = (Q, X, \delta, q_0, F)$ such that*

$$L = \sqrt[\mathsf{Suff}]{\bigcap_{q \in Q} L_1(A_q)}.$$

If, in addition, L is a regular language, then there is a finite acceptor with this property. Moreover, if $\mathsf{Pref}(L) \not\supseteq X$, the acceptor can be chosen to be strongly connected.

Proof. The statement follows from Propositions 6 and 8, Corollary 3, Remark 2 and Lemma 3. ❑

Example 4 Let $X = \{a, b\}$ and L = $\{aba^i b^2 | i \in \mathbb{N}\}$. The language L is regular and a maximal solid code (see [JY90, Yu05]). The states and transitions of the Levenshtein-Romanov acceptor for L are summarised in the following table:

State	Transition	
	a	b
ε	a	ε
a	a	ab
ab	aba	ε
$aba^i, i \in \mathbb{N}$	aba^{i+1}	$aba^i b$
$aba^i b, i \in \mathbb{N}$	aba	$aba^i b^2$
$aba^i b^2, i \in \mathbb{N}$	a	ε

Accepting states are $\{aba^i b^2 | i \in \mathbb{N}\}$. Reduction results in the following six states:

$$[\varepsilon], [a], [ab], [aba], [abab] \text{ and } [abab^2].$$

The reduced acceptor is shown in Fig. 6. ❑

Theorem 9 *Let $L \subseteq X^+$ be a non-empty solid code. For every word $w \in X^+$, the Levenshtein-Romanov acceptor for L exposes the L-decomposition of w.*

Proof. Let $\vec{w} = (n, \vec{u}, \vec{v})$ be the L-decomposition of w, where

$$u = (u_1, u_2, \ldots, u_n) \text{ and } v = (v_0, v_1, \ldots, v_n).$$

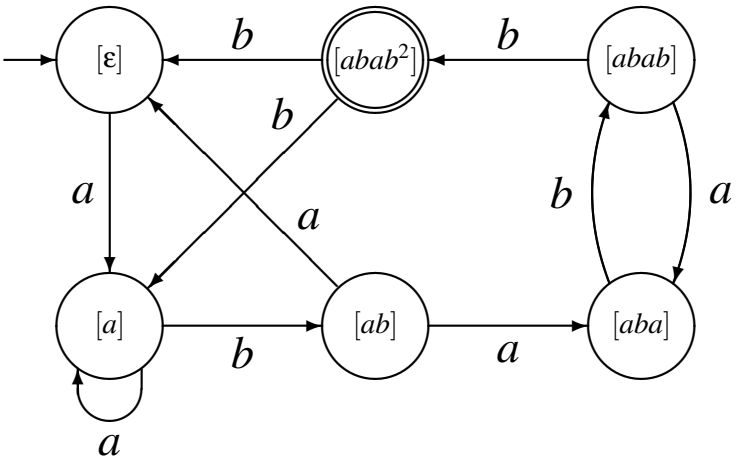


Figure 6: The reduced Levenshtein-Romanov acceptor for the solid code $\{aba^ib^2 | i \in \mathbb{N}\}$ of Example 4.

For $n = 0$, no prefix of $w = v_0$ is accepted. Let $n > 0$. Then $v_0u_1 \in L_1(A)$ and $\sigma_L(v_0u_1) = u_1 \in L$. Assume now, that $v_0u_1v_1 \cdots u_i \in L_i(A)$ for i with $1 \leq i < n$. As $u_i \in L$, one has

$$\sigma_L(v_0u_1v_1 \cdots u_i) = u_i,$$

hence

$$\sigma_L(v_0u_1v_1 \cdots u_iz) = \sigma_L(z)$$

for all $z \in X^+$ by Proposition 2. Using the properties of the decomposition,

$$\sigma_L(v_0u_1v_1 \cdots u_iz) = \sigma_L(z) \notin L$$

for every $z <_p v_iu_{i+1}$ and

$$\sigma_L(v_0u_1v_1 \cdots u_iv_iu_{i+1}) = \sigma_L(v_iu_{i+1}) = u_{i+1} \in L.$$

Thus

$$v_0u_1v_1 \cdots u_iv_iu_{i+1} \in L_{i+1}(A).$$

❑

The construction of the state-invariant transducer without look-ahead used in the proofs of Theorem 3 DIFFERS slightly from our construction of the Levenshtein-Romanov acceptor as follows: There the set of states is $\mathsf{Pref}_+(L) \cup \{\varepsilon\}$ instead of $\mathsf{Pref}(L)$. The transition from

a state q upon input x leads to $\sigma_L(qx)$ if $\sigma_L(qx) \notin L$, and to ε otherwise. Hence, in that construction, the state ε cannot be interpreted as an accepting state as it is reached by the empty word or a word which does not end on a proper prefix of L or a word which ends on a word in L. To distinguish these cases, for each state q and each input symbol x, a Mealy output $m(q,x)$ is issued, which is the empty word, if $\sigma_L(qx) \notin L$, and the decoding of qx otherwise. In our construction these situations are separated. Hence we could attach Moore outputs to the states, not the transitions, as follows: For an accepting state q, that is, $q \in L$, the output is the decoding of q; otherwise the output is the empty word. With this modification, the Levenshtein-Romanov acceptor for a finite solid code can be considered as a decoder. On the other hand, the older construction also exposes the L-decomposition of every word by issuing non-empty outputs at the ends of the words in L.

9 Reduced Fault-Tolerant Acceptors for Solid Codes

The Examples 2 and 4 lead to the assumption that for regular solid codes L the reduced Levenshtein-Romanov acceptors are finite. The results of this section show that this is indeed the case. To this end we prove in Theorem 10 a certain converse to Theorem 6.

We start with a property of left quotients of codes.

Lemma 5 *Let $L \subset X^+, L \neq \emptyset$.*

1. *L is an overlap-free prefix code if and only if $u^{[-1]}L \cap \mathsf{Pref}_+(L) = \emptyset$ for all $u \in X^*$.*

2. *L is a suffix code if and only if $u^{[-1]}L \cap X^*L = \emptyset$ for all $u \neq \varepsilon$.*

Proof.

1. Assume $v \in u^{[-1]}L \cap \mathsf{Pref}_+(L)$. Then $uv \in L$, and $\varepsilon <_p v <_p v'$ for some $v' \in L$. If $u = \varepsilon$ then $v, v' \in L$, hence L is no prefix code. If $u \neq \varepsilon$ then $uv \in L$ and $v' \in L$ overlap.

   Conversely, if L is no prefix code then $u^{[-1]}L \cap \mathsf{Pref}_+(L) \neq \emptyset$ for $u = \varepsilon$. If L is not overlap-free then $(uv)w = u(vw)$ with $u, v, w \in X^+$ and $uv, vw \in L$. Consequently, $v \in u^{[-1]}L \cap \mathsf{Pref}_+(L)$.

2. Assume $v \in u^{[-1]}L \cap X^*L$. Then $uv \in L$ and there is a $v' \in X^*$ such that $v = v'v''$ with $v'' \in L$. This implies $v'' <_s uv$ contradicting the assumption that L is a suffix code.

   If L is not a suffix code then $v <_s uv$ for some $u \in X^+$ and $v, uv \in L$, that is, $v \in u^{[-1]}L \cap X^*L$. ❑

Theorem 10 *Let L be a non-empty subset of X^+, and let A be an initially connected acceptor with $L(A) = X^*L$. The following statements hold true:*

1. *A has no useless states.*
2. *If L is an overlap-free p-infix code then*
$$L \subseteq L^{\mathrm{ol-pi}}(A) \subseteq \sqrt[\mathsf{Pref}]{X^*L}.$$
3. *If L is a solid code then $L^{\mathrm{solid}}(A) = L$.*

Proof. Let $A = (Q, X, \delta, q_0, F)$. First we prove that A has no useless states. Consider a state q of A. As A is initially connected, there is a word w such that $\delta(q_0, w) = q$. For $v \in L$ one has $wv \in X^*L$, hence $\delta(q_0, wv) = \delta(q, v) \in F$. Therefore, q is not useless.

For any word $w \in X^*$, let
$$M_w = \bigcup_{u \in \mathsf{Suff}(w), u \neq \varepsilon} u^{[-1]}L.$$
If $q = \delta(q_0, w)$, one has
$$L(A_q) = w^{[-1]}(X^*L) = X^*L \cup M_w = L(A) \cup M_w.$$

Now assume that L is an overlap-free p-infix code. We show that $L \subseteq L_1(A_q)$ for $q \in Q$. Let $\delta(q_0, w) = q$ and assume that $v \in L \setminus L_1(A_q)$. Then there is a $v' \in L_1(A_q) = X^*L \cup M_w$ such that $\varepsilon <_p v' <_p v$, that is, $v' \in \mathsf{Pref}_+(L)$. By Lemma 5.1 $v' \notin M_w$. Thus $v' = uv''$ where $u \in X^*$ and $v'' \in L$ and we obtain $uv''u' = v$ for some $u' \in X^+$, a contradiction to $X^*LX^+ \cap L = \emptyset$.

The inclusion $\bigcap_{q \in Q} L_1(A_q) \subseteq L_1(A)$ is obvious, and Proposition 1 yields $L_1(A) = \sqrt[\mathsf{Pref}]{X^*L}$.

Finally, if L is a solid code, it is not only an overlap-free p-infix code, but also a suffix code, hence $L = \sqrt[\mathsf{Suff}]{X^*L}$ and, therefore, Item 2 yields
$$L \subseteq L^{\mathrm{solid}}(A) \subseteq \sqrt[\mathsf{Suff}]{X^*L} = L.$$

❑

The inclusions in Theorem 10.2 can be proper. We derive an example.

Example 5 Let $L = \{abc, bc\} \subseteq \{a,b,c\}^*$. Then L is an overlap-free p-infix code. Let $L(A) = \{a,b,c\}^*L = \{a,b,c\}^*bc$. The equations in the proof of Theorem 10 show $L(A) \subseteq L(A_q) \subseteq L(A) \cup \{c\}$ for all $q \in Q$. Thus $a^*bc \subseteq L_1(A_q)$ for all $q \in Q$. Moreover, $ca^*bc \subseteq L_1(A)$ and $c \in L_1(A_q)$ if $q = f(q_0, b)$. This yields $L \subset \bigcap_{q \in Q} L_1(A_q) \subset L_1(A) = \sqrt[\mathsf{Pref}]{\{a,b,c\}^*L}$.

Observe that the prefix code $L' := \sqrt[\mathsf{Pref}]{\{a,b,c\}^*L}$ is p-infix but not overlap-free. This follows from $\{a,b,c\}^*bc \cap \mathsf{Pref}_+(L') = \emptyset$ and $cbc \in L'$. ❑

Note that the sets M_w as defined in the proof of Theorem 10 determine the states in the following sense.

Lemma 6 *Let $L \subseteq X^+$ be a non-empty suffix code. For $w \in X^*$ let M_w as defined in the proof of Theorem 10. Then $M_w \cap X^*L = \emptyset$. Hence, for $v, w \in X^*$, one has $v \equiv_{X^*L} w$ if and only if $M_v = M_w$.*

Proof. Suppose $u \in M_w \cap X^*L$. Then $u = rs$ with $s \in L$ and $tu \in L$ for some suffix $t \in X^+$ of w. Hence, $s <_s tu$. Since L is a suffix code, this is impossible. As $v^{[-1]}(X^*L) = X^*L \cup M_v$ and $w^{[-1]}(X^*L) = X^*L \cup M_w$, one has $v \equiv_{X^*L} w$ if and only if $M_v = M_w$. ❑

It is well known that, for any given language L, the reduced acceptor A of L can be built incrementally based on the equivalence classes of the Nerode equivalence with respect to L. To obtain the states and to define the transitions of A, one enumerates the words in X^* according to the length-lexicographical order $\cong_{\mathrm{ll}}$. The states are denoted by representatives of the equivalence classes of words. When all words up to and including w have been considered, one has an acceptor, partial or total,

$$A^{(w)} = (Q^{(w)}, X, \delta^{(w)}, q_\varepsilon, F^{(w)}).$$

One starts with $A^{(\varepsilon)}$, where $Q^{(\varepsilon)} = \{q_\varepsilon\}$, $\delta^{(\varepsilon)}$ is nowhere defined, and where $F^{(\varepsilon)}$ either contains q_ε or is empty depending on whether $\varepsilon \in L$ or not.

Assume that $A^{(v)}$ has been built. If $A^{(v)}$ is total, then the process ends with $A = A^{(v)}$; moreover, let $A^{(w)} = A^{(v)}$ for all $w \in X^*$ with $v \cong_{\mathrm{ll}} w$. Otherwise, the process continues with the word w following v in the length-lexicographical order by building $A^{(w)}$ from $A^{(v)}$. Let $w = ux$ with $x \in X$. Then $u \cong_{\mathrm{ll}} v$. If $\delta^{(v)}(\delta^{(v)}(q_\varepsilon, u),)$ is defined then $A^{(w)} = A^{(v)}$. Otherwise, if w is equivalent to some $r \cong_{\mathrm{ll}} v$ then define $Q^{(w)} =$

$Q^{(v)}, F^{(w)} = F^{(v)}$ and

$$\delta^{(w)}(q,y) = \begin{cases} \delta^{(v)}(q_\varepsilon, r), & \text{if } q = \delta^{(v)}(q_\varepsilon, u) \text{ and } y = x, \\ \delta^{(v)}(q,y), & \text{otherwise, when defined,} \\ \text{undefined}, & \text{otherwise,} \end{cases}$$

for all $q \in Q^{(w)}$ and $y \in X$. If w is not equivalent to some such r, let q_w be a new state, $Q^{(w)} = Q^{(v)} \cup \{q_w\}$,

$$F^{(w)} = \begin{cases} F^{(v)}, & \text{if } w \notin L, \\ F^{(v)} \cup \{q_w\}, & \text{if } w \in L, \end{cases}$$

and

$$\delta^{(w)}(q,y) = \begin{cases} q_w, & \text{if } q = \delta^{(v)}(q_\varepsilon, u) \text{ and } y = x, \\ \delta^{(v)}(q,y), & \text{otherwise, when defined,} \\ \text{undefined}, & \text{otherwise,} \end{cases}$$

for all $q \in Q^{(w)}$ and $y \in X$. The construction is summarised as follows:

Proposition 7 *For every language L, the procedure above converges to the reduced acceptor A for L in the following sense:*

1. *If $A^{(w)}$ is total for some $w \in X^*$ then $A = A^{(v)}$ for every $v \in X^*$ with $w \cong_{\mathrm{ll}} v$.*
2. *For every $w \in X^*$, if $A^{(w)}$ is partial then the states in $Q^{(w)}$ represent classes of the Nerode equivalence, $\delta^{(w)}$ is defined by the multiplication of these classes by symbols on the right, and $F^{(w)}$ consists of exactly those states in $Q^{(w)}$ which represent classes contained in L. Moreover, for every word $v \in X^*$ with $w <_{\mathrm{ll}} v$, $A^{(w)}$ is partial subacceptor of the partial or total acceptor $A^{(v)}$, and for some such v, this inclusion is proper.*

Moreover, assume that the following two properties are decidable for all $v, w \in X^*$:

- *Is $w \in L$?*
- *Is $v^{[-1]}L = w^{[-1]}L$?*

Then the process is effective. In particular, if L is regular, then the finite reduced acceptor for L is obtained in finitely many steps.

This is, of course, well-known, but rarely stated in these terms. For the present paper we use these ideas applied to the the special case when the language under consideration has the form X^*L where L is a solid code. Using Theorem 10 and Proposition 7, for any solid code L, one can build a reduced acceptor equivalent to the Levenshtein-Romanov acceptor for L.

Theorem 11 *Let $L \subseteq X^+$ be a non-empty solid code. The reduced Levenshtein-Romanov acceptor for L is determined by the procedure above. Moreover, if L is given constructively as a regular language[6], then the reduced Levenshtein-Romanov acceptor for L can be computed from the description of L.*

Proof. One builds the reduced acceptor for X^*L. When L is constructively regular, the two properties above in Proposition 7 are decidable. ❑

The construction of the reduced Levenshtein-Romanov acceptor for a given regular solid code as outlined above is highly inefficient. For detailed complexity analyses of similar algorithms and related ones we refer to [BPR10, CR94, CH97, CHL01, CR02, Wat95] and literature cited there. The construction outlined above can most likely be improved. A simple upper bound on the time complexity of the task of constructing the reduced Levenshtein-Romanov acceptor for a given regular solid code can be obtained from Theorem 10 using well-known automaton theoretic algorithms:

Remark 5 *Let $L \subseteq X^+$ be a non-empty solid code, given as an initially connected deterministic finite acceptor $A = (Q, X, \delta, q_0, F)$ with $L = L(A)$. Construct the non-deterministic acceptor $A' = (Q, X, \delta', q_0, F)$ with*

$$\delta'(q,x) = \begin{cases} \{\delta(q,x)\}, & \text{if } q \neq q_0, \\ \{\delta(q,x)\} \cup \{q_0\}, & \text{if } q = q_0, \end{cases}$$

where $q \in Q$ and $x \in X$. Then $L(A') = X^*L$. Next, build an initially connected deterministic acceptor $A'' = (Q'', X, \delta'', q_0'', F'')$ such that $L(A'') = L(A') = X^*L$. Finally, one reduces A''. Let $n = |Q|$ and $m = |X|$. Constructing A' takes $O(m)$ steps. One has $|Q''| \leq 2^n$, and building A'' takes at most $O(m \cdot 2^n)$ steps. Reducing A'' requires at most $O(m \cdot |Q''| \cdot log|Q''|) \leq O(m \cdot 2^n \cdot n)$ steps.

[6] This includes finite automata, regular expressions or one-sided linear grammars; it excludes, for example, descriptions by means for which the regularity might be undecidable or descriptions obtained by non-constructive existence proofs.

Thus, the reduced Levenshtein-Romanov acceptor for L can by obtained in time[7] $O(m+m\cdot 2^n+m\cdot n\cdot 2^n)=O(m\cdot n\cdot 2^n)$.

As an acceptor A with $L(A)$ a solid code has quite specific properties [HS11] it seems likely that the bound stated in Remark 5 can be improved significantly.

Acknowledgement

This research was supported in part by the Natural Sciences and Engineering Council of Canada.

References

[AC75] Alfred V. Aho and Margaret J. Corasick. Efficient string matching: an aid to bibliographic search. *Commun. ACM*, 18:333–340, 1975.

[Bal02] Vladimir B. Balakirsky. Block codes for asynchronous data transmission designed from binary trees. *Comput. J.*, 45(2):243–248, 2002.

[BP85] Jean Berstel and Dominique Perrin. *Theory of codes*, volume 117 of *Pure and Applied Mathematics*. Academic Press, Orlando, 1985.

[BPR10] Jean Berstel, Dominique Perrin, and Christophe Reutenauer. *Codes and automata*, volume 129 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, Cambridge, 2010.

[CH85] Renato M. Capocelli and Christoph M. Hoffmann. Algorithms for factorizing and testing subsemigroups. In Alberto Apostolico and Zvi Galil, editors, *Combinatorial algorithms on words. (Proceedings of the NATO Advanced Research Workshop on Combinatorial Algorithms on Words held at Maratea, Italy, June 18-22, 1984).*, pages 59–81, Berlin, 1985. Springer-Verlag.

[7] We assume that the cost for all operations including storage, retrieval, comparisons and so on is independent of n and m.

[CH97] Maxime Crochemore and Christophe Hancart. Automata for matching patterns. In *Handbook of Formal Languages*, volume 2, pages 399–462. Springer-Verlag, 1997.

[CHL01] Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithmique du texte.* Vuibert, Paris, 2001. English translation: *Algorithms on strings*, Cambridge University Press, Cambridge, 2007.

[CR94] Maxime Crochemore and Wojciech Rytter. *Text algorithms.* Oxford Univ. Press, Oxford, 1994.

[CR02] Maxime Crochemore and Wojciech Rytter. *Jewels of stringology*. World Scientific, Singapore, 2002.

[DJKM12] Mark Daley, Helmut Jürgensen, Lila Kari, and Kalpana Mahalingam. Relativized codes. *Theor. Comput. Sci.*, 429:54–64, 2012.

[FRS07] Henning Fernau, Klaus Reinhardt, and Ludwig Staiger. Decidability of code properties. *Theor. Inform. Appl.*, 41(3):243–259, 2007.

[Gle61] Yuri V. Glebskii. Coding by means of finite automata. *Dokl. Akad. Nauk. SSSR*, 141:1054–1057, 1961. in Russian. English translation: *Soviet Physics Dokl.* 6 (1992), 1037–1039.

[GP72] Ferenc Gécseg and Istvan Peák. *Algebraic theory of automata*. Akadémiai Kiadó, Budapest, 1972.

[HB98] W. Cary Huffman and Richard A. Brualdi. *Handbook of Coding Theory*. Elsevier Science Inc., New York, NY, USA, 1998.

[Hea00] Tom Head. Relativized code concepts and multi-tube DNA dictionaries. In Cristian Calude and Gheorghe Puaun, editors, *Finite Versus Infinite*, Discrete mathematics and theoretical computer science, pages 175–186. Springer, 2000.

[Hof84] Christoph M. Hoffmann. A note on unique decipherability. In Michal Chytil and Václav Koubek, editors, *Mathematical Foundations of Computer Science 1984; Proceedings, 11th Symposium; Praha, Czechoslovakia; September 3–7, 1984*, volume 176 of *Lecture Notes in Computer Science*, pages 50–63, Berlin, 1984. Springer-Verlag.

[HS11] Yo-Sub Han and Kai Salomaa. Overlap-free languages and solid codes. *Int. J. Found. Comput. Sci.*, 22(5):1197–1209, 2011.

[HW06] Yo-Sub Han and Derick Wood. Overlap-free regular languages. In Danny Z. Chen and Der-Tsai Lee, editors, *Computing and combinatorics. 12th annual international conference, COCOON 2006, Taipei, Taiwan, August 15–18, 2006. Proceedings.*, volume 4112 of *Lecture Notes in Computer Science*, pages 469–478, Berlin, 2006. Springer.

[IJST91] Masami Ito, Helmut Jürgensen, Huei-Jan Shyr, and Gabriel Thierrin. Outfix and infix codes and related classes of languages. *J. Comput. System Sci.*, 43:484–508, 1991.

[JK97] Helmut Jürgensen and Stavros Konstantinidis. Codes. In Rozenberg and Salomaa [RS97a], pages 511–607.

[JK05] Helmut Jürgensen and Stavros Konstantinidis. Worst case redundancy of solid codes. In Do Long Van and Masami Ito, editors, *The Mathematical Foundation of Informatics, Proceedings of the Conference, Hanoi, Vietnam, 25–28 October 1999*, pages 85–94, Singapore, 2005. World Scientific.

[JK06] Helmut Jürgensen and Stavros Konstantinidis. (Near-)inverses of sequences. *Int. J. Comput. Math.*, 83(2):203–222, 2006.

[JKK01] Helmut Jürgensen, Masashi Katsura, and Stavros Konstantinidis. Maximal solid codes. *J. Autom. Lang. Comb.*, 6(1):25–50, 2001.

[JKL04] Helmut Jürgensen, Stavros Konstantinidis, and Nguyen Huong Lâm. Asymptotically optimal low-cost solid codes. *J. Autom. Lang. Comb.*, 9(1):81–102, 2004.

[JKM06] Natasa Jonoska, Lila Kari, and Kalpana Mahalingam. Involution solid and join codes. In Oscar H. Ibarra and Zhe Dang, editors, *Developments in Language Theory, 10th International Conference, DLT 2006, Santa Barbara, CA, USA, June 26-29, 2006, Proceedings*, volume 4036 of *Lecture Notes in Computer Science*, pages 192–202, Berlin, 2006. Springer-Verlag.

[JKM08] Natasa Jonoska, Lila Kari, and Kalpana Mahalingam. Involution solid and join codes. *Fundam. Inform.*, 86(1-2):127–142, 2008.

[Jür99] Helmut Jürgensen. Syntactic monoids of codes. *Acta Cybernet.*, 14(1):117–133, 1999.

[Jür09] Helmut Jürgensen. Markers and deterministic acceptors for non-deterministic languages. *J. Autom. Lang. Comb.*, 14(1):33–62, 2009.

[Jür11] Helmut Jürgensen. Marks of changes in sequences. In George Venkov, Ralitza Kovacheva, and Vesela Pasheva, editors, *Applications of mathematics in engineering and economics. Proceedings of the 37th international conference (AMEE '11), Sozopol, Bulgaria, June 8–13, 2011.*, pages 319–330, Melville, NY, 2011. American Institute of Physics (AIP).

[Jür13] Helmut Jürgensen. Automata for codes. In Stavros Konstantinidis, editor, *Implementation and application of automata. 18th international conference, CIAA 2013, Halifax, NS, Canada, July 16–19, 2013. Proceedings*, pages 2–15, Berlin, 2013. Berlin: Springer.

[Jür14] Helmut Jürgensen. Towards a systematic theory of codes. Proceedings of ICRAM 2014, 2014.

[JY90] Helmut Jürgensen and Shyr-Shen Yu. Solid codes. *J. Inf. Process. Cybern.*, 26(10):563–574, 1990.

[KM06] Lila Kari and Kalpana Mahalingam. Involution solid codes. In Junghuei Chen, Natasa Jonoska, and Grzegorz Rozenberg, editors, *Nanotechnology: Science and Computation*, Natural Computing Series, pages 137–146. Springer, 2006.

[KY10] Stavros Konstantinidis and Joshua Young. f-words and binary solid codes. *Journal of Automata, Languages and Combinatorics*, 15(3/4):269–283, 2010.

[Lâm01] Nguyen Huong Lâm. Finite maximal solid codes. *Theor. Comput. Sci.*, 262(1-2):333–347, 2001.

[Lâm03] Nguyen Huong Lâm. Completing comma-free codes. *Theor. Comput. Sci.*, 301(1-3):399–415, 2003.

[Lev61] Vladimir I. Levenshtein. Self-adaptive automata for decoding messages. *Dokl. Akad. Nauk. SSSR*, 141:1320–1323, 1961. in Russsian. English translation: *Soviet Physics Dokl.* 6 (1961), 1042–1045.

[Lev62] Vladimir I. Levenshtein. The inversion of finite automata. *Dokl. Akad. Nauk. SSSR*, 147:1300–1303, 1962. in Russian. English translation: *Soviet Physics Dokl.* 7 (1963), 1081–1084.

[Lev64a] Vladimir I. Levenshtein. Decoding automata, invariant with respect to the initial state. *Probl. Kibern.*, 12:125–136, 1964. in Russian.

[Lev64b] Vladimir I. Levenshtein. Some properties of coding and self-adjusting automata for decoding messages. *Probl. Kibern.*, 11:63–121, 1964. in Russian.
An English translation is available from the Clearinghouse for Federal Scientific and Technical Information, U. S. Department of Commerce, under the title *Problems of Cybernetics, Part II,* document AD 667 849; it was prepared as document FTD-MT-24-126-67 by the Foreign Technology Division, U. S. Air Force,
German translation: Über einige Eigenschaften von Codierungen und von selbstkorrigierenden Automaten zur Decodierung von Nachrichten, in [LKT66], pp. 96–163.

[Lev70] Vladimir I. Levenshtein. On the maximum number of words in codes without overlaps. *Problemy Peredachi Informatsii*, 6(4):88–90, 1970. in Russsian. English translation: *Problems Inform. Transmission* 6 (1973) 4, 355–357.

[Lev04] Vladimir I. Levenshtein. Combinatorial problems motivated by comma-free codes. *J. Combin. Des.*, 12(3):184–196, 2004.

[LKT65] Aleksej A. Ljapunow, Wilhelm Kämmerer, and Helmut Thiele, editors. *Probleme der Kybernetik*, volume 8. Akademie-Verlag, Berlin, 1965.

[LKT66] Aleksej A. Ljapunow, Wilhelm Kämmerer, and Helmut Thiele, editors. *Probleme der Kybernetik*, volume 7. Akademie-Verlag, Berlin, 1966.

[Mar62] Aleksandr A. Markov. Non-recurrent coding. *Probl. Kibern.*, 8:169–186, 1962. in Russian. German translation: Nicht rekurrente Codierung in [LKT65], pp. 154–175.

[PT90] Mario Petrich and Gabriel Thierrin. The syntactic monoid of an infix code. *Proc. Amer. Math. Soc.*, 109:865–873, 1990.

[Rom66] O.T. Romanov. On invariant decoding automata without look-ahead. *Probl. Kibern.*, 17:233–236, 1966. in Russian.

[RS97a] Grzegorz Rozenberg and Arto Salomaa, editors. *Handbook of Formal Languages*, volume 1. Springer-Verlag, Berlin, 1997.

[RS97b] Grzegorz Rozenberg and Arto Salomaa, editors. *Handbook of Formal Languages*, volume 2. Springer-Verlag, Berlin, 1997.

[Shy01] Huei-Jan Shyr. *Free Monoids and Languages*. Hon Min Book Company, Taichung, third edition, 2001.

[Sta69] Peter H. Starke. *Abstrakte Automaten*. Deutscher Verlag der Wissenschaften, Berlin, 1969. in German. English translation: *Abstract Automata*, North-Holland, Amsterdam, 1972.

[SY90] Huei-Jan Shyr and Shyr-Shen Yu. Solid codes and disjunctive domains. *Semigroup Forum*, 41:23–37, 1990.

[Thi73] Gabriel Thierrin. The syntactic monoid of a hypercode. *Semigroup Forum*, 6:227–231, 1973.

[Thi81] Gabriel Thierrin. Hypercodes, right convex languages and their syntactic monoids. *Proc. Am. Math. Soc.*, 83:255–258, 1981.

[Val77] Erich Valkema. Syntaktische Monoide und Hypercodes. *Semigroup Forum*, 13:119–126, 1976/77.

[Wat95] Bruce W. Watson. *Taxonomies and Toolkits of Regular Language Algorithms. Proefschrift (doctoral thesis),*. PhD thesis, Technische Universiteit Eindhoven, Eindhoven, 1995.

[Yu05] Shyr-Shen Yu. *Languages and Codes*. Tsang Hai Book Publishing Co., Taichung, Taiwan, 2005.