Fault-Tolerant Acceptors for Solid Codes
(Extended Abstract)

Helmut Jürgensen∗ Ludwig Staiger†

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

Solid codes were introduced as strongly regular codes in 1964 by Levenshtein [16] and later called codes without overlap in [18]. The term solid codes seems to appear in [22] for the first time. A combinatorial characterization of solid codes is provided in [15]. Many properties of solid codes are summarized in [14]. Further information about solid codes is provided in the full version of the present paper.

The fault-tolerance afforded by a solid code $L$ can be summarized 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 characterize the class of solid codes.

For finite solid codes an automaton characterization was published in 1964 by Levenshtein [16] and in 1966 by Romanov [19]. The characterization 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 acceptance does not depend on the intial state – every state can be used as the initial state (see [14]).

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 characterization 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 characterization can be reduced to an equivalent finite automaton with equivalent properties. This is by no means as obvious as it seems. Our characterization 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. Intuitively, though not quite truly, the acceptors involved in our construction are also 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.

∗Department of Computer Science, The University of Western Ontario, London, Ontario, Canada N6A 5B7. Email: hjj@csd.uwo.ca
†Institut für Informatik, Martin-Luther-Universität Halle-Wittenberg, von-Seckendorff-Platz 1, 06120 Halle, Germany. Email: staiger@informatik.uni-halle.de

Presented at the 14th Mons Days of Theoretical Computer Science, Université catholique de Louvain, Belgium, September 11 - 14, 2012
As general references we use: [20, 23] for the theories of formal languages and automata; [2, 3, 14, 21, 24] for the theory of codes.

2 Notation and Basic Notions

As general references we use: [20, 23] for the theories of formal languages and automata; [2, 3, 14, 21, 24] for the theory of codes. By \( \mathbb{N} \) and \( \mathbb{N}_0 \) we denote the sets of positive and non-negative integers, respectively. An alphabet is a set. Let \( X \) be an alphabet. The set of all words over \( X \), including the empty word \( \varepsilon \), is denoted by \( X^* \). To exclude the empty word we write \( X^+ \), that is, \( X^+ = X^* \setminus \{ \varepsilon \} \). A language over \( X \) is a subset of \( X^* \). In our context, the case when the alphabet \( X \) is empty or a singleton set usually leads to trivial and clumsy exceptions. In the sequel we assume 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 \( w \in X^* \), \( \text{Pref}(w) = \{ u \mid u \in X^*, w \in uX^* \} \) is the set of prefixes of \( w \). The set of proper prefixes of \( w \) is \( \text{Pref}_p(w) = \text{Pref}(w) \setminus \{ \varepsilon, w \} \). For \( L \subseteq X^* \), let \( \text{Pref}(L) = \bigcup_{w \in L} \text{Pref}(w) \) and \( \text{Pref}_p(L) = \bigcup_{w \in L} \text{Pref}_p(w) \). One defines suffixes and infixes of words analogously. Thus \( \text{Suff}(w) = \{ u \mid u \in X^*, w \in X^*u \} \) is the set of suffixes of \( w \), and \( \text{Inf}(w) = \{ u \mid u \in X^*, w \in X^*uX^* \} \) is the set of infixes of \( w \). The sets of proper suffixes and proper infixes of \( w \) are \( \text{Suff}_p(w) = \text{Suff}(w) \setminus \{ \varepsilon, w \} \) and \( \text{Inf}_p(w) = \text{Inf}(w) \setminus \{ \varepsilon, w \} \), respectively. For \( r \in \{ \text{Pref}, \text{Suff} \} \) and \( L \subseteq X^+ \), the \( r \)-root of \( L \) is the set \( \sqrt{L} = \{ w \mid w \in L, r(w) \cap L = \emptyset \} \).

For languages \( L, L' \subseteq X^* \), \( L \circ L' \) is the shuffle product of \( L \) and \( L' \).

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>prefix code</td>
<td>( LX^+ \cap L = \emptyset )</td>
<td>( \mathcal{L}_p )</td>
</tr>
<tr>
<td>suffix code</td>
<td>( X^+ L \cap L = \emptyset )</td>
<td>( \mathcal{L}_s )</td>
</tr>
<tr>
<td>infix code</td>
<td>( \text{Inf}_p(L) \cap L = \emptyset )</td>
<td>( \mathcal{L}_i )</td>
</tr>
<tr>
<td>bifix code</td>
<td>( L \in \mathcal{L}_p \cap \mathcal{L}_s )</td>
<td>( \mathcal{L}_b )</td>
</tr>
<tr>
<td>overlap-free</td>
<td>( \text{Pref}_p(L) \cap \text{Suff}_p(L) = \emptyset )</td>
<td>( \mathcal{L}_{ol} )</td>
</tr>
<tr>
<td>solid code</td>
<td>( L \in \mathcal{L}<em>s \cap \mathcal{L}</em>{ol} )</td>
<td>( \mathcal{L}_{solid} )</td>
</tr>
<tr>
<td>p-infix code</td>
<td>( X^+ LX^+ \cap L = \emptyset )</td>
<td>( \mathcal{L}_{pi} )</td>
</tr>
<tr>
<td>s-infix code</td>
<td>( X^+ LX^* \cap L = \emptyset )</td>
<td>( \mathcal{L}_{si} )</td>
</tr>
<tr>
<td>comma-free code</td>
<td>( X^+ LX^+ \cap L^2 = \emptyset )</td>
<td>( \mathcal{L}_{comma-free} )</td>
</tr>
<tr>
<td>hypercode</td>
<td>( (L \circ X^+) \cap L = \emptyset )</td>
<td>( \mathcal{L}_h )</td>
</tr>
</tbody>
</table>

To keep statements simple, we allow for \( L \) to be empty in all these cases. A language \( L \subseteq X^+ \) is a prefix codes or a suffix code, if and only if \( \text{Pref}_p(L) = L \) or \( \text{Suff}_p(L) = L \), respectively. Overlaps are called “borders” in some parts of the literature. For details about these classes of languages, see [14]. The relations between these classes are shown in Fig. 1.

![Figure 1: Relations between the language or code classes considered.](image-url)
**Definition 1** Let $A = (Q, X, \delta, q_0, F)$ and $i \in \mathbb{N}$. 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 $w_1, w_2, \ldots, w_{i-1} \in \text{Pref}_+(w)$ 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 state a well-known automaton theoretic characterization of prefix codes (see [2, 14]).

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

**Definition 2** Let $P$ be a property of words in $X^+$. A $P$-decomposition of a word $w \in X^+$ is a construct $w = (n, u, v)$ with the following properties: (1) $n \in \mathbb{N}_0$; (2) $u = (u_1, u_2, \ldots, u_n)$ with $u_1, u_2, \ldots, u_n \in X^+$, each satisfying $P$; (3) $v = (v_0, v_1, \ldots, v_n)$ with $v_0, v_1, \ldots, v_n \in X^+$ such that no $v_i$ has an infix satisfying $P$; (4) $w = v_0 u_1 v_1 u_2 v_2 \cdots u_n v_n$.

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 and, then attempt to invert the encoding, possibly correcting errors. In general, a word may have any finite number of decompositions according to $P$. The first part of this process is modelled by an acceptor as follows.

**Definition 3** Let $A$ be an acceptor, let $P$ be a property of words in $X^+$, let $w \in X^+$, and let $w = (n, u, v)$ be a $P$-decomposition of $w$. Let $\text{Pref}_P(w) = \{v_0 u_1 v_1 u_2 v_2 \cdots u_i \mid i = 1, 2, \ldots, n\}$. The acceptor $A$ exposes the $P$-decomposition $w$ if and only of the following conditions are satisfied: (1) $t \in L(A)$ for all $t \in \text{Pref}_P(w)$; (2) $t \notin L(A)$ for all $t \in \text{Pref}(w) \setminus \text{Pref}_P(w)$.

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.

**Definition 4** [16, 18] 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 $w = (n, u, v)$ with $n \in \mathbb{N}_0$, $u = (u_1, u_2, \ldots, u_n)$ 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 $i = 0, 1, \ldots, n$, $\text{Inf}(v_i) \cap L = \emptyset$. An $L$-decomposition is a special kind of $P$-decomposition according to Definition 2. For every $L \subseteq X^+$, every word in $X^+$ has at least one $L$-decomposition.

**Definition 5** A solid code over $X$ is a language $L \subseteq X^+$ such that every word in $X^+$ has a unique $L$-decomposition.

**Theorem 1** [15] Solid codes as defined in Definitions 4 and 5 are the same.

There is a subtle difference between the two definitions of solid codes which becomes apparent when one relativizes 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 $M \subseteq X^+$.

By the proposed standard relativization technique of [9], the relativized versions of the two definitions are not equivalent in general.

## 3 State-Invariant Decoders for Finite Solid Codes

For finite solid codes a characterization in terms of transducers is given in [19]. This work is presented in more general terms in Chapter 11 of [14]. Here we provide a brief summary of these results. A finite deterministic transducer is a construct $A = (Q, X, Y, \delta, \mu)$ with the following properties and interpretation: $Q$ is a finite non-empty set of states; $X$ is the input alphabet and $Y$ is the output alphabet; $X$ and $Y$ are finite and non-empty; $\delta : Q \times X \to Q$ is the transition function; $\mu : Q \times X \to Y^*$ is the output function. The behaviour of a transducer on input words is defined by sequentiality.
Definition 6 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 $A = (Q, X, Y, \delta, \mu)$ with the following properties: (1) for all $q \in Q$ and all $v \in L$, $\mu(q, v) = f^{-1}(v)$; (2) for all $q \in Q$, all $v \in L$ and all $u \in \text{Pref}_+(v)$, $\mu(q, u) = \epsilon$.

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 $w$.

Theorem 2 [14, 19] 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 $f$.

4 Levenshtein-Romanov Mapping

In our construction of acceptors for solid codes, the following mapping, due to Levenshtein [16] and Romanov [19], is essential. This mapping has been re-introduced and used in unrelated contexts independently as explained below.

Definition 7 Let $X$ be an alphabet and let $L \subseteq X^+$. The mapping $\sigma_L : X^+ \rightarrow X^*$ defined by $\sigma_L(w) = \max_{\leq_s} (\text{Suff}(w) \cap \text{Pref}(L))$ is called the Levenshtein-Romanov mapping for $L$.

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

1. $w \in \text{Pref}(L)$ if and only if $\sigma_L(w) = w$;
2. if $w' \leq_s w$ then $\sigma_L(w') \leq_s \sigma_L(w)$;
3. if $\sigma_L(w) \leq_s w'$ then $\sigma_L(w) = \sigma_L(w')$;
4. $\sigma_L(ww') = \sigma_L(\sigma_L(w)w')$;
5. $\sigma_L(\epsilon) = \epsilon$;
6. if $L$ is an overlap-free prefix code, then $\sigma_L(wwv) = \sigma_L(v)$ for all $w \in L$ and $v \in X^+$;
7. if $L$ is an overlap-free prefix code, then $\sigma_L(vwv) = \sigma_L(v)$ for all $w \in L$ and $v \in X^+$;
8. if $L$ is an s-infix code, then $\sigma_L(vwv) = w$ for all $w \in L$ and $v \in X^*$.

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 [1]. Various versions of this algorithm are presented and analysed in [3, 5, 6, 7, 8]. In that 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 [4, 11] 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 basic ideas of Aho-Corasick automata are also found in Levenshtein’s 1964 paper on properties of coding and self-adjusting automata [17]. Some parts of Proposition 2 are proved more or less implicitly in [5, 14, 16, 19].

5 Fault-Tolerant Acceptors for Solid Codes

We now consider fault-tolerant acceptors for solid codes. The underlying idea is derived from Definition 6, Theorem 2 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 is 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. *A priori* it is not at all obvious that this should be true.

We follow the structure of Proposition 1, the characterisation of prefix codes by acceptors: In Theorem 3 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 4 we state that every solid code $L$ defines an acceptor $A$ such that the solid code $L'$ defined by $A$ coincides with $L$.

**Theorem 3** 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 $\bigcap_{q \in Q} L_1(A_q)$ is a solid code.

In the statements of Theorem 3 the intersection expresses a kind of state invariance; taking the suffix root can be interpreted as avoiding look-ahead. Moreover, one proves that taking the suffix root is essential.

By Theorem 3, 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 [16, 19], 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 2. 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, the language $L$ would have to satisfy additional assumptions.

**Definition 8** Let $L \subseteq X^+$. The Levenshtein-Romanov acceptor for $L$ is the acceptor $A = (Q, X, \delta, q_0, F)$ defined as follows: (1) $Q = \text{Pref}(L)$; (2) $q_0 = \epsilon$; (3) $F = L$; (4) $\delta(q, x) = \sigma_L(qx)$ for $q \in Q$ and $x \in X$.

**Proposition 3** Let $L \subseteq X^+$. Let $A = (Q, X, \delta, q_0, F)$ be the Levenshtein-Romanov acceptor for $L$. The following statements hold true: (1) for all $q \in Q$ and all $x \in X^*$, one has $\delta(q, w) = \sigma_L(qw)$; (2) $L \subseteq L(A) \subseteq X^*L; (3)$ if $X^+LX^+ \cap L = \emptyset$ then $L(A) = X^*L; (4)$ if $X^+LX^+ \cap L = \emptyset$ then $\text{Pref}(L) \cap X^*L = L$.

The converse of of Proposition 3(4) is not true in general.

**Corollary 1** Let $L \subseteq X^+$, and let $A$ be the Levenshtein-Romanov acceptor for $L$. For $x \in \{\pi, si, i, \text{solid}\}$, if $L \in \mathcal{L}_x$ then $L(A) = X^*L$.

**Corollary 2** Let $L \subseteq X^+$, and let $A$ be the Levenshtein-Romanov acceptor for $L$. If $\text{Pref}(L) \cap X^*L = L$ and $L$ is regular then also $L(A)$ is regular.

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

**Remark 1** The condition $\text{Pref}(L) \cap X^*L = L$ does not imply that $L$ is a code. Consider the language $L = \{ab, abab\}$. Then $\text{Pref}(L) = \{\epsilon, a, ab, aba, abab\}$. Hence $\text{Pref}(L) \cap X^*L = L$. However, $L$ is not a code. The same language also satisfies $X^+LX^+ \cap L = \emptyset$.

**Proposition 4** Let $L \subseteq X^+$ be an overlap-free prefix code. Let $A$ be the Levenshtein-Romanov acceptor for $L$. Then, $L_i(A_w) = L_i(A)$ and $L(A_w) \setminus \epsilon = L(A)$ for all $w \in \text{Pref}(L)$ and all $i \in \mathbb{N}$. 
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 $\varepsilon$.

**Proposition 5** Let $L \subseteq X^+$ and $A$ 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 $q \in \text{Pref}(L) = Q$; (2) if $L$ is an infix code, then $L \subseteq L_1(A)$; (3) if $L$ is a solid code, then $L \subseteq L_1(A_q)$ for all $q \in \text{Pref}(L) = Q$.

By combining the statements of Propositions 4 and 5 one obtains the following property of the Levenshtein-Romanov construction.

**Theorem 4** Let $L \subseteq X^+$ be a solid code, and let $A$ be the Levenshtein-Romanov acceptor for $L$. Then $L = \bigcap_{q \in \text{Pref}(L)} L_1(A_q)$. 

**Corollary 3** A language $L \subseteq X^+$ is a solid code if and only if there is an acceptor $A = (Q, X, \delta, q_0, F)$ such that $L = \bigcap_{q \in Q} L_1(A_q)$. If, in addition, $L$ is a regular language, then there is a finite acceptor with this property.

**Theorem 5** Let $L \subseteq X^+$ be a solid code. For every word $w \in X^+$, the Levenshtein-Romanov acceptor for $L$ exposes the $L$-decomposition of $w$.

The construction of the state-invariant transducer without look-ahead used in the proofs of Theorem 2 differs slightly from our construction of the Levenshtein-Romanov acceptor as follows: There one uses Mealy outputs; those acceptors can be interpreted to use Moore outputs. It is known that Mealy and Moore automata are essentially equivalent [23].

### 6 Summary

A characterization of finite solid codes by state-invariant decoders without look-ahead is extended to arbitrary solid codes. Even after reduction the key properties of being state-invariant and of not looking ahead are preserved. In particular, a solid code which is regular as a language, has a finite state-invariant acceptor without look-ahead. A similar construction is used in pattern matching. Our results provide for a new connection between fault-tolerant decoding and pattern matching in the sense of Aho-Corasick automata.

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


Information about the Author(s)

Helmut Jürgensen
Department of Computer Science,
The University of Western Ontario,
London, Ontario, Canada N6A 5B7
E-Mail: hjj@csd.uwo.ca
WWW: http://www.csd.uwo.ca/faculty/helmut/

Ludwig Staiger
Institut für Informatik
Martin-Luther-Universität Halle-Wittenberg
Institut für Informatik
D-06099 Halle, Germany
E-Mail: ludwig.staiger@informatik.uni-halle.de
WWW: http://www.informatik.uni-halle.de/struktur/theorie/