A fast algorithm for the unique decipherability of multivalued encodings*

Renato M. Capocelli**
Dipartimento di Matematica, Università di Roma "La Sapienza", Piazzale A. Morro 2, 185 Roma, Italy

Luisa Gargano
Dipartimento di Informatica ed Applicazioni, Università di Salerno, Via Urbano II, I-84081 Baronissi (SA), Italy

Ugo Vaccaro
Dipartimento di Informatica ed Applicazioni, Università di Salerno, Via Urbano II, I-84081 Baronissi (SA), Italy

Abstract

Multivalued encodings constitute an interesting generalization of ordinary encodings in that they allow each source symbol to be encoded by more than one codeword. In this paper the problem of testing the property of unique decipherability of multivalued encodings is considered. We provide an efficient algorithm whose time complexity is $O(|C|A)$, where $|C|$ is the number of codewords and $A$ is the sum of their lengths. It is remarkable that the running time equals that of the fastest algorithms for testing the much simpler property of unique decipherability of ordinary encodings.

1. Introduction

An encoding system is called multivalued if there may be two or more codewords corresponding to the same source symbol. This paper discusses the unique

Correspondence to: U. Vaccaro, Dipartimento di Informatica ed Applic., Università di Salerno, Via Urbano II, I-84081 Baronissi (SA), Italy. Email: uv@udsab.dia.unisa.it.

* This work was supported in part by the Italian Ministry of University and Scientific Research under Project: “Algorithmi e Sistemi di Calcolo”.

** Deceased April 8, 1992.
decipherability property of multivalued encodings with special emphasis on the time and space complexity of the problem. An efficient algorithm for testing the unique decipherability of multivalued encodings is presented.

Multivalued encodings arise in many practical situations. For instance, consider the effect of noise when a sequence of symbols is transmitted over an unreliable channel. The corresponding channel output is not uniquely determined but can be any of a set of sequences, depending both on the transmitted sequence and the error pattern that has occurred. Notice that if the channel allows insertion and deletion errors, the output sequences associated to each input sequence may have different lengths. The behavior of such a channel can be modeled by means of multivalued encoding in which the set of codewords corresponding to a source symbol represents the noisy versions of the original encoding of that symbol. However, this approach can be practical only if the set of sequences associated to each source symbol is not too large. Generally speaking, one can prevent this situation by ignoring all channel output sequences having small probability of occurrence.

Another important situation that can be successfully modeled by means of multivalued encodings is the *homophonic* channel. In the homophonic channel the set of different codewords that correspond to a source symbol represents the homophons into which that symbol is encoded. The technique of homophonic substitution is an old technique used in Cryptology for converting an actual plaintext sequence in a (more) random sequence in order to increase the message entropy. Amongst the randomization techniques it seems by far the most adequate. It has been very recently reconsidered and enriched. In particular, a complete information-theoretic treatment [17] and a general universal algorithm for homophonic encoding [12] have been provided. The multivalued encoding formalism would then permit to characterize the deciphering properties of the homophonic substitution.

It is interesting to point out that multivalued encodings arise also in molecular biology. Indeed, in the biological code several groups of bases may correspond to the same amino acid. This situation is described saying that the biological code is degenerate (see [25,26] for a detailed discussion of this peculiar aspect of the biological code).

The main difficulty in trying to characterize properties of multivalued encodings comes from the fact that for multivalued encodings *unique decipherability* is not equivalent to *unique decomposability* (i.e. a code message might be parsed in terms of codewords in two different ways, both giving the same deciphering in terms of source symbols). The situation is similar, and in a certain sense complementary, to the situation that arises in the recently considered *multi-set decipherable codes*, introduced by Lempel in [18], where code messages might have different parsings in terms of codewords but every parsing must yield the same multiset of codewords and thus the same (unordered) sequence of source symbols. The nonequivalence between unique decomposability and unique decipherability implies that the extension to multivalued encodings of fundamental properties of ordinary encodings is not straightforward, neither does it appear possible to use methods employed successfully in the ordinary
encoding case (see [11] for instance). The extension and the characterization of such properties in case only substitution errors are allowed has been done in [13, 14]. The general situation that includes insertion and deletion errors has been considered in [5, 6, 9, 10, 24]. Sato [24] was the first to consider the problem of the unique decipherability and gave a decision procedure to test whether a multivalued encoding has this property. He showed that the problem of testing whether a multivalued encoding is uniquely decipherable is equivalent to the problem of deciding whether a regular set is included in a particular context-sensitive language. Using tools of formal language theory, he showed that the last problem is indeed decidable. Unfortunately, Sato did not give indications about the time and space needed to perform his test and a direct application of his method gives inefficient algorithms. An extension of Sato's result to the case in which the set of codewords associated to each source symbol is rational has been recently given by Head and Weber [15].

The main contribution of this paper is an efficient algorithm for testing the unique decipherability of finite multivalued encodings. Conceptually, our approach is close in spirit to that by Rodeh [22] and Apostolico and Giancarlo [2] in that we directly exploit the combinatorial structure of the code and this allows us to use the advanced pattern matching techniques of [1] to obtain good algorithmic performances. We remark that our result can also be seen as further evidence that the algorithmic complexity of the fastest tests for the unique decipherability property of ordinary encodings (see [16, 22]) seems to be too high, as suggested in [16]. Indeed, the same order of complexity suffices to test the more involved property of unique decipherability for multivalued encodings. We recall that the above quoted algorithms for ordinary encodings essentially test whether there exists a code message that can be parsed in two ways, thus destroying the unique decipherability. This does not suffice in case of multivalued encodings in that both decompositions could give the same deciphering. It follows that additional tests are required. Nevertheless, these additional tests can be done in such a way that the order of magnitude of the time and space complexity of our algorithm is the same as that of the algorithms presented in [2, 16, 22].

2. Notations and definitions

Let \( X \) be a finite set of symbols, denote by \( X^m \) the set of all strings obtained by concatenating \( m \) symbols of \( X \). Let \( X^+ = \bigcup_{m \geq 1} X^m \) be the set of all finite strings of elements of \( X \) and \( X^* = X^+ \cup \{ \lambda \} \), \( \lambda \) denoting the empty string. We denote by \( \ell(w) \) the length of the string \( w \), i.e., if \( w = x_1 \ldots x_m \in X^+ \), \( x_i \in X \), we have \( \ell(w) = m \).

**Definition 2.1.** Let \( A \) be a finite source alphabet, \( X \) a finite code alphabet, and \( C \subset X^* \) a finite set of strings (codewords). A multivalued encoding is a mapping \( F: A \rightarrow 2^C \) from the source alphabet \( A \) to the set of all subsets of \( C \), denoted by \( 2^C \).
Multivalued encodings have the property that each source symbol may have different encodings. For each source symbol \( a \in A \), the associated set \( F(a) \) contains all possible encodings of the symbol \( a \). It is obvious that the above definition reduces to that of ordinary encoding when sets \( F(a) \) have cardinality one for each \( a \in A \). The set \( F(a_1 \ldots a_n) \) of all possible encodings of the string of source symbols \( a_1 \ldots a_n \) is formed by all possible concatenations of codewords associated to the source symbols \( a_i \)'s; formally

\[
F(a_1 \ldots a_n) = \{w_{i_1} \ldots w_{i_n} | w_{i_1} \in F(a_1), \ldots, w_{i_n} \in F(a_n)\}.
\]

A multivalued encoding is uniquely decipherable if no two different strings of source symbols can be encoded with the same string of code symbols. More precisely we have the following definition.

**Definition 2.2.** A multivalued encoding \( F : A \rightarrow 2^C \) is uniquely decipherable if and only if for each pair \( a_1 \ldots a_n, b_1 \ldots b_m \) of strings of source symbols, \( a_1 \ldots a_n \neq b_1 \ldots b_m \in A^+ \) implies \( F(a_1 \ldots a_n) \cap F(b_1 \ldots b_m) = \emptyset \).

From Definition 2.2 it follows that the possibility of parsing a code message in different ways does not imply that the multivalued encoding is not uniquely decipherable, unlike the ordinary encoding case. In fact, if different parsings of the received code message correspond to the same string of source symbols, one can still recover the original source string. The following examples illustrate the problem.

**Example 2.3.** The multivalued encoding defined by \( F(0) = \{a, uu\} \) and \( F(1) = \{baa\} \) is not uniquely decipherable. In fact the code message \( uubuu \) belongs both to \( F(01) \) and \( F(00) \).

**Example 2.4.** Let \( A = \{0, 1\} \) be the set of source symbols, \( X = \{a, b, c\} \) be the set of code symbols and \( C = \{a, b, ac, ca, bc, cb\} \) be the set of codewords. Let the multivalued encoding \( F \) be defined by \( F(0) = \{a, ac, ca\} \) and \( F(1) = \{b, bc, cb\} \).

The code message \( acbcb \) can be parsed in terms of codewords as \( (ac)(b)(cb) \) as well as \( (ac)(bc)(b) \), and \( (a)(cb)(cb) \), however, each parsing gives the same deciphering 011. We shall show in the rest of the paper that the multivalued encoding is indeed uniquely decipherable.

Given a string \( w = x_1 \ldots x_m \in X^+ \), with \( x_i \in X \), we call the string \( p = x_1 \ldots x_j \) a prefix of \( w \) and the string \( s = x_j \ldots x_m \) a suffix of \( w \), for any \( j = 1, \ldots, m \); the empty string \( \lambda \) is both prefix and suffix of any word. Given a set of codewords \( C \) we indicate by \( \text{Suffix}(C) \) the set of all nonnull suffixes of codewords in \( C \). Moreover, we indicate by \( C^+ \) be the set of all concatenations of codewords of \( C \).

**Definition 2.5.** Given a set of codewords \( C \), a sequence of codewords \( w_1, \ldots, w_n \), with \( w_i \in C \), is called a factorization of the string of code symbols \( \beta \in C^+ \) if \( \beta = w_1 \ldots w_n s \), for some \( s \in \text{Suffix}(C) \cup \{ \lambda \} \).
Notice that a string $\beta$ may have different factorizations.

3. Unique decipherability of multivalued encodings

In this section we shall establish a necessary and sufficient condition for a multivalued encoding to be uniquely decipherable. Such a condition will give ground for the derivation of an efficient algorithm for testing the unique decipherability property. We first develop some machinery that will be useful to prove the correctness of the algorithm. A central role is played by the notion of L-sequence, a sequence of overlapping (linked) codewords. L-sequences seem quite an useful tool in the theory of variable length codes and have found wide application in characterizing fundamental properties of both ordinary and multivalued encodings [4, 5, 8].

**Definition 3.1** (Capocelli [4]). Given a set of codewords $C$, we call L-sequence with prefix $s_0$ any sequence of codewords $w_1, \ldots, w_n \in C$, $n \geq 1$, such that

$$w_1 = s_0s_1,$$

$$w_2 = s_1s_2 \quad \text{or} \quad s_1 = w_2s_2,$$

$$\vdots$$

$$w_n = s_{n-1}s_n \quad \text{or} \quad s_{n-1} = w_ns_n,$$

and denote it by $(s_0, w_1, \ldots, w_n)$. We call $s_n$, the suffix of the L-sequence.

Given an L-sequence $(s_0, w_1, \ldots, w_n)$ with suffix $s_n$, construct two sequences $\Sigma_1$ and $\Sigma_2$, hereafter referred as $\Sigma$-sequences, as follows:

(i) Put $w_1$ in $\Sigma_1$ and $w_2$ in $\Sigma_2$;

(ii) for each succeeding $w_i$, $i = 3, \ldots, n$,

if $w_{i-1} - s_{i-2}s_{i-1}$ adjoin $w_i$ to the $\Sigma$-sequence which does not contain $w_{i-1}$;

if $s_{i-2} = w_{i-1}s_{i-1}$ adjoin $w_i$ to the same $\Sigma$-sequence as $w_{i-1}$.

Given a sequence of strings $x = x_1, \ldots, x_k$, with $x_i \in X^+$, let $c(x) = c(x_1, \ldots, x_k) \in X^+$ represent the string obtained by concatenating the words in $x$, i.e.

$$c(x) = c(x_1, \ldots, x_k) = x_1 \cdots x_k.$$

Given an L-sequence $(s_0, w_1, \ldots, w_n)$ with suffix $s_n$, it is easy to see that its associated $\Sigma$-sequences satisfy one of the following two relations:

$$c(\Sigma_1, s_n) = c(s_0, \Sigma_2),$$

$$c(\Sigma_1) = c(s_0, \Sigma_2, s_n).$$
Moreover, given an L-sequence \((s_0, w_1, \ldots, w_n)\) with prefix \(s_0 = w_0 \in C\) consider the corresponding \(\Sigma\)-sequences and define the two sequences of codewords \(\Pi_1\) and \(\Pi_2\) as follows:

\[
\Pi_1 = w_0, \Sigma_2 \quad \text{and} \quad \Pi_2 = \Sigma_1 \quad \text{if (2a) holds};
\]

\[
\Pi_1 = \Sigma_1 \quad \text{and} \quad \Pi_2 = w_0, \Sigma_2 \quad \text{if (2b) holds}.
\]

Notice that \(c(\Pi_1)\) and \(c(\Pi_2)\) are both code messages, i.e. \(c(\Pi_1), c(\Pi_2) \in C^+\), and \(c(\Pi_1) = c(\Pi_2, s_n)\) holds. Therefore, by Definition 2.5, \(\Pi_1\) and \(\Pi_2\) are two different factorizations of the same string of code symbols \(c(\Pi_1) = c(\Pi_2, s_n)\). Moreover, \(\Pi_1\) and \(\Pi_2\) are uniquely determined by the L-sequence they are associated to.

**Example 3.2.** Consider the set of codewords \(C = \{aa, aab, bbab, ba\}\). We can construct the L-sequence \((s_0, w_1, w_2, w_3, w_4)\), with \(s_0 = aa, w_1 = aab, w_2 = bbab,\) and \(w_3 = w_4 = ba\) as follows:

\[w_1 = aab = (aa)(b) = s_0s_1, \quad w_2 = bbab = (b)(bab) = s_1s_2,\]

\[w_3 = bab = (ba)(b) = w_3s_3, \quad w_4 = ba = (b)(a) = s_3s_4.\]

The L-sequence has suffix \(s_4 = a\). The associated \(\Sigma\)-sequences are \(\Sigma_1 = w_1, w_3, w_4 = (aab), (ba), (ba)\) and \(\Sigma_2 = w_2 = bbab\) with \(c(\Sigma_1) = w_1w_3w_4 = aab-baba = s_0w_2s_4 = c(s_0, \Sigma_2, s_4)\). Moreover, \(\Pi_1 = \Sigma_1 = (aab), (ba), (ba)\) and \(\Pi_2 = s_0, \Sigma_2 = (aa), (bbab)\); according to Definition 2.5 both \(\Pi_1\) and \(\Pi_2\) are factorizations of \(aabbaba\).

Consider now the L-sequence \((s_0, w_1, w_2)\) with \(s_0 = aa, w_1 = aab\) and \(w_2 = ba\) given by

\[w_1 = aab = (aa)(b) = s_0s_1, \quad w_2 = ba = (b)(a) = s_1s_2.\]

The L-sequence has suffix \(s_2 = a\). One has \(\Sigma_1 = w_1 = aab\) and \(\Sigma_2 = w_2 = ba\) with \(c(\Sigma_1, s_2) = c(s_0, \Sigma_2)\). Moreover, \(\Pi_1 = s_0, \Sigma_2 = (aa), (ba)\) and \(\Pi_2 = \Sigma_1 = (aab);\) according to Definition 2.5 both \(\Pi_1\) and \(\Pi_2\) are factorizations of \(aaba\).

The following theorem is easy to derive, a full proof can be found in [4].

**Theorem 3.3** [Capocelli [4]]. Given a set of codewords \(C\), the following two conditions are equivalent:

1. there exists a code message \(\beta = w_1 \cdots w_r = v_1 \cdots v_s\) with \(w_i, v_j \in C\) for \(i = 1, \ldots, r, j = 1, \ldots, s\) and \(w_1 \neq v_1;\)

2. there exists an L-sequence of prefix \(s_0\) and suffix \(s_n\) with \(s_0, s_n \in C\).

It follows that, in order to test the unique decipherability property of multivalued encodings, one has to consider only L-sequences whose prefix is a codeword. Unless otherwise specified, all L-sequences considered in the following will have a codeword as prefix.
Define now the decoding of a codeword \( w \in C \) as \( F^{-1}(w) = \{ a \in A \mid w \in F(a) \} \). In the sequel we assume that the size of such a set is one, otherwise the encoding is trivially not unique decipherable. Moreover, define the decoding of a sequence of codewords \( w_1, \ldots, w_n \) as

\[
F^{-1}(w_1, \ldots, w_n) = F^{-1}(w_1) \ldots F^{-1}(w_n).
\]

Given an L-sequence \( (w_0, w_1, \ldots, w_n), w_0 \in C \), with associated factorizations, \( \Pi_1 \) and \( \Pi_2 \), its remainder, \( \text{Rem}(w_0, w_1, \ldots, w_n) \), is defined as

\[
\text{Rem}(w_0, w_1, \ldots, w_n) = \begin{cases} 
(\lambda, \lambda) & \text{if } F^{-1}(\Pi_1) = F^{-1}(\Pi_2) \lambda, \\
(\lambda, \lambda) & \text{if } F^{-1}(\Pi_2) = F^{-1}(\Pi_1) \lambda, \\
(\lambda, \lambda) & \text{if } F^{-1}(\Pi_1) = F^{-1}(\Pi_2), \\
\emptyset & \text{otherwise,}
\end{cases}
\]

where \( \lambda \) is a string of source symbols.

**Example 2.4 (continued).** Consider the code of Example 2.4 and the L-sequence \( (w_0, w_1, w_2, w_3, w_4) \) with \( s_0 = w_0 = a \) and suffix \( s_4 = ba \) given by

\[
w_1 = ac = (a)(c), \quad w_2 = cb = (c)(b),
\]

\[
w_3 = bc = (b)(c), \quad w_4 = ca = (c)(a).
\]

We have \( \Sigma_1 = w_1, w_3 = (ac), (bc) \) and \( \Sigma_2 = w_2, w_4 = (cb), (ca) \). The \( \Pi \)-sequences are \( \Pi_1 = w_0, \quad \Pi_2 = \Sigma_1, (ac), (bc) \) and \( \Pi_2 = \Sigma_2, (cb), (ca) \). Since \( F^{-1}(\Pi_1) = F^{-1}(ac)F^{-1}(bc) = 010 \) and \( F^{-1}(\Pi_2) = F^{-1}(ca)F^{-1}(bc) = 01 \), one has \( \text{Rem}(a, ac, cb, bc, ca) = (0, \lambda) \).

Given \( k \geq 1 \) and \( p \in \text{Suffix}(C) \), define \( A_k(p) \) as the set of all the L-sequences \( (w_0, w_1, \ldots, w_n) \) with prefix \( w_0 \in C \) such that

1. \( 1 \leq n \leq k; \)
2. \( (w_0, w_1, \ldots, w_n) \) has suffix \( s_n = p. \)

Finally, define \( A(p) = \bigcup_{k \geq 1} A_k(p) \) and \( S = |\text{Suffix}(C)| \). For each \( p \in \text{Suffix}(C) \) it holds

\[
A(p) \neq \emptyset \quad \text{if and only if} \quad A_S(p) \neq \emptyset.
\]  

(3)

The “only if” implication follows from the fact that for \( n > S = |\text{Suffix}(C)| \) any L-sequence \( (w_0, w_1, \ldots, w_n) \) must have two suffixes \( s_i \) and \( s_j \) such that \( s_i = s_j. \) This is, the
L-sequence has the following structure:

\[
\begin{align*}
\ldots & \quad \ldots \\
 W_i &= S_i_{-1} S_i & \text{or} & & S_i_{-1} &= W_i S_i, \\
 W_{i+1} &= S_i S_{i+1} & \text{or} & & S_i &= W_{i+1} S_{i+1} \\
 \ldots & \quad \ldots \\
 W_j &= S_j_{-1} S_j & \text{or} & & S_j_{-1} &= W_j S_j, \\
 W_{j+1} &= S_j S_{j+1} & \text{or} & & S_j &= W_{j+1} S_{j+1} \\
 \ldots & \quad \ldots 
\end{align*}
\]

Therefore, from the L-sequence \((w_0, w_1, \ldots, w_k)\), one can obtain a shorter L-sequence
\((w_0, w_1, \ldots, w_i, w_{j+1}, \ldots, w_\ell)\) \((w_0, w_1, \ldots, w_k)\) if \(j = n\) having the same suffix \(p\); and so on until one gets an L-sequence in \(A_\ell(p)\).

The following theorem holds.

**Theorem 3.4.** A multivalued encoding \(F: A \rightarrow 2^C\) is uniquely decipherable if and only if for each source symbol \(a \in A\) and for each codeword \(w \in F(a)\) the remainders of all L-sequences in the set \(A(w)\) assume the same value \((a, \lambda)\).

**Proof.** Suppose \(F\) not uniquely decipherable. Then there exists a shortest code message \(\beta = v_1 \ldots v_k = w_1 \ldots w_m\) with \(F^{-1}(v_1, \ldots, v_k) \neq F^{-1}(w_1, \ldots, w_m)\). Without loss of generality, suppose that \(v_1\) is a proper prefix of \(w_1\) and that \(v_k\) is a suffix of \(w_m\). From Theorem 3.3 it follows that there exists an L-sequence whose prefix and suffix are codewords. In particular, from the proof of Theorem 3.3 given in [4] it follows that there exists a L-sequence of prefix \(w_0 = v_1\) and suffix \(s_k = v_k\), and associated \(\Pi\) sequences \(\Pi_1 = v_1, \ldots, v_{k-1}\) and \(\Pi_2 = w_1, \ldots, w_m\). By definition of \(A(v_k)\) such an L-sequence belongs to \(A(v_k)\) and cannot have remainder equal to \((F^{-1}(v_k), \lambda)\), otherwise \(F^{-1}(v_1, \ldots, v_k) = F^{-1}(\Pi_1)F^{-1}(v_k) = F^{-1}(\Pi_2) = F^{-1}(w_1, \ldots, w_m)\).

Suppose now that \(F\) is uniquely decipherable. Let \(w \in F(a)\) and \((w_0, w_1, \ldots, w_n)\) be an L-sequence in \(A(w)\). Consider the associated sequences \(\Pi_1\) and \(\Pi_2\). It holds \(c(\Pi_1) = c(\Pi_2, w)\). Moreover, \(F^{-1}(\Pi_1) = F^{-1}(\Pi_2, w)\), otherwise we would get \(F^{-1}(\Pi_1)F^{-1}(\Pi_2, w) \in A^+\), \(F^{-1}(\Pi_1) \neq F^{-1}(\Pi_2, w)\), and \(F(F^{-1}(\Pi_1)) = c(\Pi_1) = c(\Pi_2, w) = F(F^{-1}(\Pi_2, w))\) contradicting the hypothesis of unique decipherability.

Finally, noticing that \(F^{-1}(\Pi_2, w) = F^{-1}(\Pi_2) a\), from the definition of remainder, we get: \(\text{Rem}(w_0, w_1, \ldots, w_n) = (a, \lambda)\). \(\square\)

Since sets \(A(w)\) are generally infinite, the above theorem does not give an effectively testable condition for the unique decipherability of multivalued encoding. The following lemma is useful to obtain a finite condition for unique decipherability.
Lemma 3.5. If for each \( p \in \text{Suffix}(C) \) the remainders of all elements in the set \( A_{S+1}(p) \) assume the same value, then for each \( p \in \text{Suffix}(C) \) the remainders of all elements in the set \( A(p) \) assume the same value.

Proof. In the Appendix. \( \square \)

From Theorem 3.4 and Lemma 3.5 one immediately obtains the following theorem.

**Theorem 3.6.** A multivalued encoding \( F \) is uniquely decipherable if and only if for each source symbol \( a \) and for each codeword \( w \in F(a) \), the remainders of all \( L \)-sequences in the set \( A_{S+1}(w) \) take the same value \((a, \lambda)\).

**Example 2.3 (continued).** Consider the multivalued encoding \( F \) of Example 2.3. One has that \( S = 6\), \((aa, aab, baa) \in A_2(aa) \subseteq A_7(aa) \) and \( \text{Rem}(aa, aab, baa) = (1, \lambda) \). Since \( F^{-1}(aa) \neq 0 \), by Theorem 3.6 the encoding \( F \) is not uniquely decipherable.

**Example 2.4. (continued).** Consider the multivalued encoding \( F \) of Example 2.4. Since \( S = 7 \), we must consider the sets \( A_8(p) \), for \( p \in \text{Suffix}(C) \). We find that

\[
A_8(bc) = A_8(ac) = A_8(cb) = A_8(ca) = \emptyset,
\]

\[
A_8(c) = \{ (x_0, x_0, c, \ldots, cx_i, cx_j, \ldots, cx_i, x_i) : x_i \in \{a, b\}, 0 \leq j \leq i, 0 \leq i \leq 3 \},
\]

\[
A_8(a) = \{ (x_0, x_0, c, \ldots, cx_i, x_j) : x_i = a, x_j \in \{a, b\}, 0 \leq j \leq i - 1, 0 \leq i \leq 3 \},
\]

\[
A_8(b) = \{ (x_0, x_0, c, \ldots, cx_i, x_j) : x_i = b, x_j \in \{a, b\}, 0 \leq j \leq i - 1, 0 \leq i \leq 3 \}.
\]

Therefore, for each \( L \)-sequence \( (w_0, w_1, \ldots, w_n) \)

\[
\text{Rem}(w_0, w_1, \ldots, w_n) = \begin{cases} 
(\lambda, \lambda) & \text{if } (w_0, w_1, \ldots, w_n) \in A_8(c), \\
(0, \lambda) & \text{if } (w_0, w_1, \ldots, w_n) \in A_8(a), \\
(1, \lambda) & \text{if } (w_0, w_1, \ldots, w_n) \in A_8(b).
\end{cases}
\]

Since \( a \in F(0) \) and \( b \in F(1) \), \( F \) is uniquely decipherable by Theorem 3.6.

**Remark (Multivalued encoding and McMillan's inequality).** It is well known that if an ordinary encoding with set of codewords \( C \subseteq X^* \) is uniquely decipherable then it satisfies the McMillan’s inequality \( \sum_{w \in C} |X|^{-\ell(w)} \leq 1 \) [21]. This inequality is important from the practical point of view since it implies that to encode the output of a given source there is no loss of optimality if one uses prefix codes only, that, on the other hand, are easy to generate and to decode. The above example shows that such a result does not hold for multivalued encodings. Indeed, the multivalued encoding considered in Example 2.4 has codeword set \( C = \{a, b, ac, bc, ca, cb\} \) and

\[
\sum_{w \in C} |X|^{-\ell(w)} = \sum_{w \in C} 3^{-\ell(w)} = 10/9 > 1.
\]
It is also possible to show that the multivalued encoding $F_m$ defined by
\begin{align*}
F_m(0) &= \{ a, ac^i, c^i a : i = 1, \ldots, m \}, \\
F_m(1) &= \{ a, bc^i, c^i b : i = 1, \ldots, m \},
\end{align*}
is uniquely decipherable (by applying Theorem 3.6) and $\sum_{w \in F_m(\emptyset) \cup F_m(1)} 3^{-\ell(w)}$ approaches $\frac{4}{3}$ as $m$ increases. An interesting open problem is that of finding a generalization of McMillan’s inequality for uniquely decipherable multivalued encodings.

4. An efficient algorithm for testing the unique decipherability of multivalued encodings

In this section we shall provide an algorithm to test whether a multivalued encoding is uniquely decipherable. The algorithm is based on Theorem 3.6 and its time and space complexities are $O(|C| \Delta)$, where $\Delta$ is the sum of the codeword lengths.

Roughly speaking, the algorithm operates by examining all the possible ambiguous sequences of length not larger than $|\text{Suffix}(C)| + 1$ that can be constructed using the codeword set $C$; notice that $\Delta = \sum_{w \in C} \ell(w) \geq |\text{Suffix}(C)|$. Recall that we assume in the set of source symbols $\mathcal{A}$ there are no two symbols which are encoded with the same codeword.

In the algorithm we use two variables $\text{error}(p)$ and $\text{Rem}(p)$. The variable $\text{error}(p)$ is binary: $\text{error}(p)$ is initially set to 0 and remains 0 as long as all the L-sequences with suffix $p$ that have been so far constructed by the algorithm have the same remainder different from 0; $\text{error}(p)$ is 1 if either the algorithm has constructed an L-sequence with suffix $p$ and remainder 0 or it has constructed two L-sequences with suffix $p$ and different remainders. It is obvious that if for some $p \in C$ it holds $\text{error}(p) = 1$ then the encoding is not uniquely decipherable.

The variable $\text{Rem}(p)$ is defined only if $\text{error}(p) = 0$. In such a case $\text{Rem}(p)$ is equal to the remainder common to all L-sequences with suffix $p$ so far considered.

Initially, for each $p \in \text{Suffix}(C)$ we put $\text{error}(p) = 0$ and $\text{Rem}(p) = (\ast, \ast)$, where $\ast$ is a special symbol matching every string.

In Step 1 of the algorithm the possible L-sequences $(v, w)$, with $w = vp$ of prefix $v \in C$ and suffix $p$ are constructed. If $F^{-1}(v) = F^{-1}(w)$ then $\text{Rem}(p) = (\lambda, \lambda)$ and $\text{error}(p)$ is left equal to 0. If $F^{-1}(v) \neq F^{-1}(w)$ then $\text{error}(p) = 1$. In both cases, in order to expand the ambiguous sequences with suffix $p$, we put $p$ in queue $Q$. Since we queue $p$ for each value of $\text{error}(p)$ only once, we update to $T$ (true) the boolean variable $\text{queue}(p, \text{error}(p))$.

Each time Step 2 is executed a suffix $p$ is taken from the queue $Q$, the L-sequences with suffix $p$ are expanded, if possible, by one codeword and the new resulting suffixes are considered. Call $q$ a new suffix, if $\text{error}(p) = 1$ then $\text{error}(q) = 1$. If $\text{error}(p) = 0$, the remainder of the expanded L-sequences is computed and compared with $\text{Rem}(q)$. If the remainder of the expanded L-sequence is $\emptyset$ or else it does not match $\text{Rem}(q)$ then $\text{error}(q)$ is 1. In the other cases $\text{error}(q)$ remains 0. The remainder of the expanded
L-sequence, \( \text{newrem}(p, v, q) \), is obtained as follows. Let \( \text{Rem}(p) = (x, y) \) (where either \( x = \lambda \) or \( y = \lambda \)) and let \( v \) be the added codeword, that is the codeword such that \( p = vq \) or \( pq = v \). Consider the sequences \( x', y' \in A^* \) such that \( x = zx' \) and \( yF^{-1}(v) = zy' \), where \( z \) is the longest common prefix between \( x \) and \( yF^{-1}(v) \). Set

\[
\text{newrem}(p, v, q) = \begin{cases} (x', y') & \text{if } p = vq, \\ (y', x') & \text{if } pq = v. \end{cases}
\]  

(4)

If either \( x' \neq \lambda \neq y' \) or \( \text{Rem}(q) \neq \text{newrem}(p, v, q) \) then set \( \text{error}(q) = 1 \); otherwise set \( \text{Rem}(q) = \text{newrem}(p, v, q) \).

The following algorithm formalizes the above reasoning.

**Algorithm**

for \( p \in \text{Suffix}(C) \) do

\( \text{error}(p) = 0; \)

\( \text{Rem}(p) = (*,*) \) [\( * \) is a special symbol matching every string];

\( \text{queue}(p, 0) = \text{queue}(p, 1) = F \) [i.e. false];

**Step 1:**

for \( v \in C \) do

for \( w \in C \) do

if \( v = wp \) and \( p \in C \) then halt; [\( \text{Rem}(p) \neq (F^{-1}(p), \lambda) \) and the encoding is not UD].

if \( v = wp \) and \( F^{-1}(v) \neq F^{-1}(w) \) then \( \text{error}(p) = 1; Q = p; \text{queue}(p, 1) = T; \)

if \( v = wp \) and \( F^{-1}(v) = F^{-1}(w) \) then \( \text{Rem}(p) = (\lambda, \lambda); Q = p, \text{queue}(p, 0) = T, \)

**Step 2:**

while \( Q \) is not empty do

\( p = Q; \)

if (there exists \( v \in C \) such that \( v = pq \) or \( p = vq \)) then

if \( \text{error}(p) = 1 \) then

\( \text{error}(q) = 1; \)

if \( \text{queue}(q, 1) = F \) then \( Q = q; \text{queue}(q, 1) = T; \)

if \( q \in C \) then halt; [the encoding is not UD].

if \( \text{error}(p) = 0 \) then

let \((x, y) = \text{newrem}(p, v, q)\)

\[
\begin{cases} x \neq \lambda \text{ and } y \neq \lambda & \text{do } \text{error}(q) = 1; \\ \text{case } \text{Rem}(q) \text{ is still } (*,*) \text{ or is } (x, y) & \text{do } \text{Rem}(q) = (x, y); \\ \text{Rem}(q) \neq (x, y) & \text{do } \text{error}(q) = 1; \end{cases}
\]

if \( \text{queue}(q, \text{error}(q)) = F \) then \( Q = q; \text{queue}(q, \text{error}(q)) = T; \)

if \( q \in C \) and (\( \text{error}(q) = 1 \) or \( \text{Rem}(q) \neq (F^{-1}(q), \lambda) \)) then halt [the encoding is not UD];

halt [the encoding is UD].
In order to prove the correctness of the proposed algorithm we need the following result.

**Lemma 4.1.** Let \( S = |\text{Suffix}(C)| \). For each \( p \in \text{Suffix}(C) \) it holds:

(a) If \( \text{error}(p) = 1 \) then there exist either an \( L \)-sequence \( (w_0, w_1, \ldots, w_n) \in A_{S+1}(p) \) such that \( \text{Rem}(w_0, w_1, \ldots, w_n) = \emptyset \) or two \( L \)-sequences \( (w_0, w_1, \ldots, w_n) \) and \( (v_0, v_1, \ldots, v_m) \) in \( A_{S+1}(p) \) such that \( \text{Rem}(w_0, w_1, \ldots, w_n) \neq \text{Rem}(v_0, v_1, \ldots, v_m) \).

(b) If \( \text{error}(p) = 0 \) and \( Q \) is empty then for each \( L \)-sequence \( (w_0, w_1, \ldots, w_n) \in A_{S+1}(p) \) (if any) it holds \( \emptyset \neq \text{Rem}(w_0, w_1, \ldots, w_n) = \text{Rem}(p) \).

**Proof.** The proof is given in the Appendix. \( \square \)

The algorithm stops in three cases:

1. When \( w \in C \) with \( \text{error}(w) = 1 \). The encoding is not UD. The correctness follows from (a) of Lemma 4.1 and Theorem 3.6.

2. When \( w \in C \) with \( \text{Rem}(w) \neq (F^{-1}(w), \lambda) \). The encoding is not UD. The correctness follows from Theorem 3.6, since there exists at least an \( L \)-sequence in \( A_{S+1}(w) \) with remainder different from \( (F^{-1}(w), \lambda) \).

3. When the queue \( Q \) is empty. The encoding is UD. In order to prove the correctness, notice that when \( Q \) empties, for any given \( p \in \text{Suffix}(C) \), all \( L \)-sequences in \( A_{S+1}(p) \) have been considered. Moreover, the above case (2) has never occurred. That is, \( \text{Rem}(w) = (F^{-1}(w), \lambda) \) for each \( w \in C \). From (b) of Lemma and Theorem 3.6 one gets that the encoding is UD.

Therefore, the algorithm correctly tests whether a multivalued encoding is uniquely decipherable.

We now prove that the time complexity of the algorithm is \( O(|C| \Delta) \), where \( \Delta = \sum_{w \in C} \ell(w) \). By using the suffix tree of the set of codewords \( C \), as described in [22] (with an added field for each suffix \( p \), to store the value of \( \text{error}(p) \)), all the operations involving comparisons between codewords require \( O(|C| \Delta) \) time, see for instance Rodeh [22].

In order to complete the proof, it remains to show that the operations involving \( \text{rems} \) and \( \text{newrems} \) (i.e. the storing of the values obtained from the \( \text{rems} \) of the various suffixes and the comparisons between a \( \text{newrem} \) and the stored value of the \( \text{rem} \)) can be done in \( O(|C| \Delta) \). The \( \text{rems} \) of the various prefixes can be stored by means of a prefix tree. A prefix tree is a labeled tree where the root represents the empty string \( \lambda \) and each node is represented by the string of labels on the path from the root to it. In our case, if a node represents the string \( x \) then for each suffix \( p \) with remainder \( (x, \lambda) \) or \( (\lambda, x) \) there is a pointer from \( p \) to \( x \) and a pointer from \( x \) to \( p \).

Consider first the case that element \( p \) currently taken from the queue \( Q \) has remainder \( (x, \lambda) \) (resp. \( (\lambda, x) \)), and that \( \text{newrem}(p, v, q) = (xa, \lambda) \) (resp. \( (\lambda, xa) \)) for some \( a \in A \). In such a case the only thing to do is to move from node \( x \) to its son \( xa \) and, depending on the step of the algorithm, either insert the new element \( \text{Rem}(q) = \text{newrem}(p, v, q) \) or to check if the pointer from \( x \) to \( xa \) and the pointer from \( q \) coincide (i.e. if
A fast algorithm for multivalued encodings

both point to \( xa \); if they do it remains to check if \( Rem(q) \) and \( \text{newrem}(p, v, q) \) are both of the form \( (xa, \lambda) \) or \( (\lambda, xa) \).

Consider now the remaining case, i.e. the remainder of \( p \) is \( (ax, \lambda) \) (resp. \( (\lambda, ax) \)) and \( \text{newrem}(p, v, q) = (x, \lambda) \) (resp. \( (\lambda, x) \)). We need to locate in the tree the node \( x \). In order to perform this operation, we can slightly modify the tree in the following way. When we insert the string \( a_1a_2(u_1, u, EA) \) as son of the node \( a_1 \) (i.e. the node represented by the string \( a_1 \)) we also insert in the tree the string \( u_2 \) as son of the root and a pointer from \( a_1a_2 \) to \( a_2 \); in general, when we insert the string \( a_1 \ldots a_r \) (as son of \( a_1 \ldots a_{r-1} \)) we also insert a pointer from it to \( a_2 \ldots a_r \) as follows: following the pointer from \( a_1 \ldots a_{r-1} \) to \( a_2 \ldots a_{r-1} \) we insert the node \( a_2 \ldots a_r \) (if it does not exist) and then the pointer from \( a_1 \ldots a_r \) to \( a_2 \ldots a_r \). Since the algorithm never looks for the remainder of a suffix more than \( |C| \) times and recalling that \( |\text{Suffix}(C)| \leq A \), we get the desired result.

**Theorem 4.2.** The algorithm tests the unique decipherability of a multivalued encoding in \( O(|C|A) \) time.

**Acknowledgments**

The authors wish to express their gratitude to the anonymous referee for the careful reading of the paper and the many suggestions that helped in improving its readability.

**Appendix**

**Proof of Lemma 3.5.** Let us suppose that the hypothesis of the Lemma is satisfied for each \( p \in \text{Suffix}(C) \) and denote by \( Rem(p) \) the remainder of the L-sequences in \( A_{S+1}(p) \). We show by induction that for each \( k > S \) if

\[
\text{for all } p \in \text{Suffix}(C) \quad (w_0, w_1, \ldots, w_n) \in A_k(p) \Rightarrow Rem(w_0, w_1, \ldots, w_n) = Rem(p)
\]

then it holds

\[
\text{for all } q \in \text{Suffix}(C) \quad (v_0, v_1, \ldots, v_m) \in A_{k+1}(q) \Rightarrow Rem(v_0, v_1, \ldots, v_m) = Rem(q)
\]

Let \( (v_0, v_1, \ldots, v_{k+1}) \) be an L-sequence in \( A_{k+1}(q) \) \( - A_k(q) \), \( q \in \text{Suffix}(C) \). Two cases can be distinguished:

\[
v_{k+1} = s_k s_{k+1} \quad \text{or} \quad s_k = v_{k+1} s_{k+1} \quad \text{with} \quad s_{k+1} = q.
\]

Consider the L-sequence \( (v_0, v_1, \ldots, v_k) \). Since it has prefix \( s_0 = v_0 \) and suffix \( s_k \), one gets that it belongs to \( A_k(s_k) \). From (3), it follows that there exists an L-sequence \( (w_0, w_1, \ldots, w_l) \in A_l(s_k) \subseteq A_k(s_k) \). By inductive hypothesis, all the L-sequences in \( A_k(s_k) \)
have the same remainder $\text{Rem}(s_k)$. Therefore, one has that

$$\text{Rem}(v_0, v_1, \ldots, v_k) = \text{Rem}(w_0, w_1, \ldots, w_l).$$

Since $(w_0, w_1, \ldots, w_l, v_{k+1}) \in A_S(p)$, by using again the inductive hypothesis one gets

$$\text{Rem}(v_0, v_1, \ldots, v_{k+1}) = \text{Rem}(w_0, w_1, \ldots, w_l, v_{k+1}) = \text{Rem}(p)$$

and the lemma is proved. □

**Proof of Lemma 4.1.** We prove by induction on the number of executions of the while loop, that at the end of any execution for each $q \in \text{Suffix}(C)$:

(a) If $\text{error}(q) = 1$, then either there exists an L-sequence $(w_0, w_1, \ldots, w_n) \in A_{S+1}(q)$ such that $\text{Rem}(w_0, w_1, \ldots, w_n) = \emptyset$ or there exist two L-sequences $(w_0, w_1, \ldots, w_n)$ and $(v_0, v_1, \ldots, v_m) \in A_{S+1}(q)$ such that $\text{Rem}(w_0, w_1, \ldots, w_n) \neq \text{Rem}(v_0, v_1, \ldots, v_m)$.

(b) If $\text{error}(q) = 0$, then for each L-sequence $(w_0, w_1, \ldots, w_n) \in A_{S+1}(q)$ such that no suffix of the L-sequences $(w_0, w_1, \ldots, w_m), m < n$, belongs to $Q$, $\emptyset \neq \text{Rem}(w_0, w_1, \ldots, w_n) = \text{Rem}(q)$.

It is trivial to see that the hypothesis is true at the beginning of Step 2. Suppose that it is true after any number $k' < k$ of iterations of the while loop. We prove that it is true after the $k$-th iteration.

Denote by $\text{error}^{k'}(q)$ the value of $\text{error}(q)$ after the $t$th iteration. Let $p$ be the element taken out from the queue. Consider the suffixes $q \in \text{Suffix}(C)$. If $\text{error}^{k-1}(q) = 1$ then condition (a) is satisfied by inductive hypothesis. Consider $q$ such that $\text{error}^{k-1}(q) = 0$ and $\text{error}^{k}(q) = 1$. Note that the variable $\text{error}(q)$ changes during the $k$th iteration only if there exists $v \in C$ such that $pq = v$ or $p = vq$. Three cases can occur.

(i) $\text{error}^{k-1}(p) = 1$. By inductive hypothesis $p$ satisfies a). Then either there exists $(w_0, w_1, \ldots, w_n) \in A(p)$ with $\text{Rem}(w_0, w_1, \ldots, w_n) = \emptyset$ or there exist $(w_0, w_1, \ldots, w_n)$ and $(v_0, v_1, \ldots, v_m) \in A(p)$ with $\text{Rem}(w_0, w_1, \ldots, w_n) \neq \text{Rem}(v_0, v_1, \ldots, v_m)$. Noticing that for each sequence $(u_0, u_1, \ldots, u_v) \in A(p)$ it holds $(u_0, u_1, \ldots, u_v) \in A(q)$ one gets that also $q$ satisfies (a).

(ii) $\text{error}^{k-1}(p) = 0$ and $\text{newrem}(q) = \emptyset$. Let $(w_0, w_1, \ldots, w_n) \in A_S(p)$ be an L-sequence such that $\text{Rem}(p) = \text{Rem}(w_0, w_1, \ldots, w_n) \neq \emptyset$. From (4), one has that $\text{newrem}(q) = \text{Rem}(w_0, w_1, \ldots, w_n, v) = \emptyset$. Then one gets that $q$ satisfies (a).

(iii) $\text{error}^{k-1}(p) = 0$ and $\text{newrem}(q) \neq \text{Rem}(q)$. Let $(w_0, w_1, \ldots, w_n) \in A_S(p)$ be a sequence such that $\text{Rem}(p) = \text{Rem}(w_0, w_1, \ldots, w_n) \neq \emptyset$ and let $(v_0, v_1, \ldots, v_m) \in A_S(q)$ be a sequence such that $\text{Rem}(q) = \text{Rem}(v_0, v_1, \ldots, v_m) \neq \emptyset$. From (4), one has $\text{newrem}(q) = \text{Rem}(w_0, w_1, \ldots, w_n, v) \neq \text{Rem}(q) = \text{Rem}(v_0, v_1, \ldots, v_m)$ and $q$ satisfies (a).

Finally we prove that suffixes $q$ for which $\text{error}^{k}(q) = 0$ (and therefore $\text{error}^{k-1}(q) = 0$) satisfy (b).

Let $(w_0, w_1, \ldots, w_n) \in A_{S+1}(q)$ and suppose that no suffix $s_i$ of $(w_0, w_1, \ldots, w_n), i < n$, is in $Q$ after the $k$th iteration. The suffix $s_1$ was queued during Step 1 and then dequeued at some iteration $k_1 \leq k$. Therefore suffix $s_2$, queued by the $k_1$th iteration, was
dequeued at some iteration \(k_2 \leq k\) and so on, until we obtain that \(s_{n-1}\) was dequeued at some iteration \(k_{n-1} \leq k\) and \(q\) was queued within the \(k_{n-1}\)-th iteration. Moreover, one has that if \(\text{error}^k(s_j) = 1\), for some \(j < n\), then \(\text{error}^k(s_i) = 1\) for \(i = j + 1, \ldots, n\) contradicting the assumption that \(\text{error}^k(q) = 0\). Therefore, \(\text{error}^k(s_j) = 0\) for all \(j = 1, \ldots, n\).

The following four cases can be distinguished (we remind that \(p\) denotes the suffix extracted from \(Q\) at the \(k\)th iteration).

1. \(s_j \neq p\), for each \(j = 1, \ldots, n - 1\). In this case (b) holds by inductive hypothesis.

2. \(s_j \neq p\), \(j = 1, \ldots, n - 2\) and \(s_{n-1} = p\). Then, since after the \((k-1)\)th iteration none of the suffixes \(s_j\), for \(j = 1, \ldots, n-2\) was in \(Q\), applying the inductive hypothesis on \(p\) we have that \(\text{Rem}(w_0, w_1, \ldots, w_{n-1}) = \text{Rem}(p)\). From this and from (4), we get that the sequence \((w_0, w_1, \ldots, w_{n-1}, w_n) \in A(q)\) with \(w_n = pq\) or \(p = w_d q\) satisfies \(\text{Rem}(w_0, w_1, \ldots, w_n) = \text{newrem}(p, w_n, q)\). Since \(\text{error}^k(q) = 0\), \(\text{Rem}(q) = \text{newrem}(p, w_n, q)\). Hence \(\text{Rem}(w_0, w_1, \ldots, w_n) = \text{Rem}(q)\).

3. \(s_i = p\) for some \(i \leq n - 2\) and \(s_j \neq p\), for \(j \neq i\) and \(j \leq n - 1\). The \(L\)-sequence \((w_0, w_1, \ldots, w_{i+1})\) satisfies above case 2, therefore, \(\text{Rem}(w_0, w_1, \ldots, w_{i+1}) = \text{newrem}(s_{i+1}) = \text{Rem}(s_{i+1})\). Moreover, since \(s_{i+1} \notin Q\), i.e., was queued and dequeued at some iteration less than \(k\), and \(s_{i+1} \neq p\), we have that there exists \((v_0, v_1, \ldots, v_m) \in A(s_{i+1})\) with all suffixes already dequeued and then different from \(p\). Since \(\text{error}^k(s_{i+1}) = 0\), by inductive hypothesis \(\text{Rem}(v_0, v_1, \ldots, v_m) = \text{Rem}(s_{i+1})\). Because all the \(s_j\), \(j > i\), have been dequeued, by using the inductive hypothesis on \((v_0, v_1, \ldots, v_m, w_{i+2}, \ldots, w_n) \in A(q)\) one gets \(\text{Rem}(q) = \text{Rem}(v_0, v_1, \ldots, v_m, w_{i+2}, \ldots, w_n, v)\).

4. There exist \(i, j\) with \(i < j \leq n - 1\) such that \(s_i = s_j = p\) and \(s_i \neq p\), for \(l < i\) and \(l > j\). The \(L\)-sequence \((w_0, w_1, \ldots, w_i)\) satisfies case 1 (since \(s_1, \ldots, s_{i-1}\) are all different from \(p\)); the \(L\)-sequence \((w_0, w_1, \ldots, w_j)\) satisfies case 2 (since \(s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_{j-1}\) are all different from \(p\)). Therefore, since \(\text{error}^k(p) = 0\), it holds

\[
\text{Rem}(w_0, w_1, \ldots, w_j) = \text{Rem}(w_0, w_1, \ldots, w_i, w_{i+1}, \ldots, w_j) = \text{Rem}(p).
\]

Expanding both the above \(L\)-sequences with \(w_{j+1}, \ldots, w_n\) they maintain an equal remainder, i.e.

\[
\text{Rem}(w_0, w_1, \ldots, w_i, w_{j+1}, \ldots, w_n) = \text{Rem}(w_0, w_1, \ldots, w_j, w_{j+1}, \ldots, w_n).
\]

Moreover, \((w_0, w_1, \ldots, w_i, w_{j+1}, \ldots, w_n)\) satisfies case 3 (since \(s_i = p\) and \(s_i \neq p\) for \(j \neq i\) and \(j \leq n - 1\)), therefore,

\[
\text{Rem}(w_0, w_1, \ldots, w_i, w_{j+1}, \ldots, w_n) = \text{Rem}(q).
\]

Combining the last two equalities, we get

\[
\text{Rem}(w_0, w_1, \ldots, w_n) = \text{Rem}(w_0, w_1, \ldots, w_i, w_{j+1}, \ldots, w_n) = \text{Rem}(q).
\]
References