Generalized factorizations of words and their algorithmic properties

Juhani Karhumäki a,*, Wojciech Plandowski b,2, Wojciech Rytter c

a Turku Centre for Computer Science and Department of Mathematics, Turku University, Finland
b Turku Centre for Computer Science and Department of Mathematics, Turku University, Finland
c Instytut Informatyki, Uniwersytet Warszawski, Banacha 2, 02-097 Warszawa, Poland, and
Department of Computer Science, University of Liverpool, UK

Abstract

We formalize the notion of a factorization of a word, a so-called $\mathcal{F}$-factorization, introduced in [7] when solving some open problems on word equations. We show that most of the factorizations considered in the literature fit well into that framework, and in particular that central algorithmic problems, such as the uniqueness or the synchronizability, remain polynomial time solvable for an important and large class of $\mathcal{F}$-factorizations, namely for regular $\mathcal{F}$-factorizations.

© 1999 Elsevier Science B.V. All rights reserved.

Keywords: Factorizations of words; Regular factorizations

1. Introduction

One of the fundamental notions of words is that of a factorization. It allows to decompose a word $w$ into a sequence of its consecutive subwords: $w = w_1 \ldots w_n$. Or dually (sub)words $w_i$ allow to build a word $w$ as their products. Typically, the subwords $w_i$ are taken from a fixed language $F$, yielding to a notion of an $F$-factorization of a word, cf. [3, 9].

Occasionally more general notions of factorizations of words has been needed. Indeed, in [7] it was crucial to consider more complicated factorizations of words in order to solve some open problems on word equations. In such factorizations, called $\mathcal{F}$-factorizations in [7], the identity $w = w_1 \ldots w_n$ defines a factorization of $w$ only if the sequence $(w_1, \ldots, w_n)$ has a "property $\mathcal{F}$". Consequently, $\mathcal{F}$-factorizations are global as a contrast to $F$-factorizations which are local in a sense that any sequence of factors

* Corresponding author. E-mail: karhumak@cs.utu.fi.
1 Supported by Academy of Finland under grant 14047.
2 On leave from Instytut Informatyki, Uniwersytet Warszawski, Banacha 2, 02-097 Warszawa, Poland.
is a factorization. Clearly, each $F$-factorization is an $\mathcal{F}$-factorization the property being “each $w_i$ belongs to $F$”.

Factorizations of words are closely related to factorizations of free monoids, cf. [2, 9]. For the latter ones also others than $F$-factorizations have been widely considered. Our motivation, however, is to consider different ways of factorizing single words as was essential in considerations in [7]. According to our knowledge no attempts to uniformly formalize such a notion has been made.

The goal of this paper is two-fold. First we want to formalize the above intuitive notion of an $\mathcal{F}$-factorization, and moreover to show that many important and natural ways of decomposing words fit into this formalism. Second, we show that several algorithmic results on $F$-factorizations can be extended to a wide class of $\mathcal{F}$-factorizations. These together, we believe, make the notion of an $\mathcal{F}$-factorization well motivated and natural.

Whenever a factorization of words is defined several natural questions arise: is it (i) uniquely deciphering, (ii) complete or (iii) synchronizing. In the case of $F$-factorizations the property (i) characterizes when $F$ is a code. Similarly, the notion (iii) is used to define synchronous codes, cf. [2]. Each of problems (i)–(iii) is known to be polynomial time solvable for $F$-factorizations with $F$ being finite, cf. [1, 2, 11]. For example, the unique decipherability problem is NL-complete, cf. [10], and is equivalent via log-space reductions to the graph accessibility problem, GAP for short.

We show that problems (i) and (iii) remain polynomial time solvable for quite a large extension of $F$-factorizations, namely for $\mathcal{F}$-factorizations where the factors are taken from regular languages and the way they are joined together is determined by another regular language. We call such $\mathcal{F}$-factorizations, which, as we shall see, cover at least most of the usually considered factorizations, regular. On the other hand, the completeness problem for regular $\mathcal{F}$-factorizations is P-SPACE-complete. If regular languages are replaced by context-free languages we obtain $\mathcal{F}$-factorizations with undecidable properties. Finally, we conclude with some open questions.

2. $\mathcal{F}$-factorizations

Let $\Sigma$ and $I = \{1, \ldots, k\}$ be disjoint alphabets. An $\mathcal{F}$-factorization $\mathcal{F} = (L, L_1, \ldots, L_k)$ over $\Sigma$ is given by languages $L \subseteq I^*$, and $L_1, \ldots, L_k \subseteq \Sigma^*$. An $\mathcal{F}$-factorization of a word $w$ is a decomposition

$$w = w_1 w_2 \ldots w_m,$$

where $w_i \in L_i$ and $i_1 i_2 \ldots i_m \in L$.

Hence, factors are words in languages $L_1, \ldots, L_k$ and the language $L$ shows how the words are composed from their factors.

An $\mathcal{F}$-factorization is regular if the languages $L$ and $L_i$ are regular. Such a factorization may have deterministic or nondeterministic representation depending on whether the languages are given by deterministic finite automata (DFA for short) or nondeter-
ministic finite automata (NFA for short), respectively. An \( \mathcal{F} \)-factorization is context-free if the languages \( L \) and \( L_i \) are context-free, usually given by context-free grammars.

Regular factorizations cover the most natural factorizations. In the following we consider several examples to support this view.

**Example 2.1.** Clearly, the simplest factorization divides a word into separate letters. Then we set \( L = 1^+ \) and \( L_1 = \Sigma \). We call this factorization trivial. Another natural factorization divides a word into blocks of the same letters. Then \( \Sigma = \{ a_i : i \in I \} \),

\[
L = 1^+ - \bigcup_{i \in I} i^* i i^* \quad \text{and} \quad L_i = a_i^+ \quad \text{for} \quad i \in I.
\]

This factorization is called here the block factorization.

**Example 2.2.** \( \mathcal{F} \)-factorizations cover also factorizations which are defined in a semigroup \( F^+ \) for a set of words \( F \), in particular which are defined by codes. Then \( L = 1^+ \) and \( L_1 = F \). Such a factorization is usually called an \( F \)-factorization.

It follows that the problems which are studied here are not of a smaller complexity than the corresponding ones for codes.

**Example 2.3.** The identity \( \{ a, b \}^* = (a^* b)^* a^* \) proposes an \( \mathcal{F} \)-factorization where \( L = 1^+ 2 \) and \( L_1 = a^* b, L_2 = a^* \). Since the expression on the right-hand side is unambiguous it follows that each binary word has a unique \( \mathcal{F} \)-factorization. Note also that the identity shows how to express \( \Sigma^* \) without using a union.

More complicated factorizations are shown in our next examples.

**Example 2.4.** For a given primitive word \( Q \) we define an \( \mathcal{F}_Q \)-factorization and \( \mathcal{F}_Q' \)-factorization as follows. Assume that the word \( w \in \Sigma^* \) is written in the form

\[
w = w_0 Q^{x_1} w_1 \ldots Q^{x_k} w_k,
\]

where, for all \( i \),

- no \( w_i \) contains \( Q^2 \) as a subword,
- \( Q \) is a proper prefix and a proper suffix of \( w_i \) for \( 0 < i < k \),
- \( Q \) is a proper suffix of \( w_0 \) or \( w_0 = 1 \),
- \( Q \) is a proper prefix of \( w_k \) or \( w_k = 1 \),
- \( x_i \geq 0 \).

An \( \mathcal{F}_Q \)-factorization of \( w \) having presentation (1) is defined as

\[
w_0, Q^{x_1}, \ldots, Q^{x_k}, w_k.
\]

In particular, for \( k = 0 \), it contains only one factor, namely the word itself. The \( \mathcal{F}_Q' \)-factorization differs from the \( \mathcal{F}_Q \)-factorization only in the sense that the factors \( Q^{x_i} \) are replaced by products of \( Q \)'s.
Lemma 2.5. Both the $\mathcal{F}_Q$-factorization and the $\mathcal{F}_Q'$-factorization are regular factorizations. Their deterministic representations have linear sizes with respect to $|Q|$.

Proof. For the $\mathcal{F}_Q$-factorization we have

$$L = (\varepsilon \cup \{1\})^* 2(\varepsilon \cup \varepsilon) \cup \mathcal{S},$$

$$L_1 = \Sigma^+ Q - \Sigma^* QQ \Sigma^*, \quad L_2 = Q^*,$$

$$L_3 = Q \Sigma^+ \cap \Sigma^+ Q - \Sigma^* QQ \Sigma^*, \quad L_4 = Q \Sigma^+ - \Sigma^* QQ \Sigma^*$$

$$L_5 = \Sigma^+ - \Sigma^* QQ \Sigma^* - \Sigma^* Q - \Sigma^*.$$

For the factorization $\mathcal{F}_Q'$ we have to change the above $L$ and $L_2$. Now we set $L = (\varepsilon \cup \{1\})^* 2(\varepsilon \cup \varepsilon) \cup \mathcal{S}$, and $L_2 = Q$.

The second sentence of the lemma follows from the standard constructions of finite automata. Indeed, the construction of linear size automata for languages $L$, $L_2$ is trivial and for languages $L_1$, $L_3$, $L_4$, $L_5$ is based on the minimal DFA $A$ which performs a pattern-matching for the pattern $QQ$, see [4]. All states in $A$ correspond to prefixes of $QQ$. We show how to use $A$ to construct a linear size DFA for the language $L_4$, the construction for other languages being similar. The language $L_4$ consists of words which start with $Q$ and do not contain $QQ$. The automaton $A_4$ for $L_4$ is the following composition of the automaton $B$ which accepts $Q$ and the automaton $A$. Let $q$ be a state in $A$ which corresponds to the word $Q$ and $q \to q'$ be an edge labeled by $a$. Then there is an edge $s \to q'$ labeled by $a$ where $s$ is the accepting state in $B$. The accepting states of $A_4$ are all those states in $A$ which do not correspond to the pattern $QQ$. The initial state is the one of $B$. \qed

A general notion of a factorization is that of an ordered factorization, see [2], used mainly in connection with factorizations of monoids. Let $I$ be a totally ordered set and let $(X_i)_{i \in I}$ be a family of sets of words over an alphabet $\Sigma$. An ordered factorization of a word $w$ is a factorization

$$w = x_1 x_2 \ldots x_n,$$

where $n \geq 1$, $x_i \in X_i$, and $j_1 \geq j_2 \geq \ldots \geq j_n$.

Observe here that the notion of an ordered factorization covers all possible factorizations of words. Indeed, each factorization can be expressed as an ordered factorization. To prove that we first define the notion of a factorization in the most general case. A factorization $F$ is a set of tuples of words $(w, x_1, \ldots, x_k)$ such that $w = x_1 \ldots x_k$ and the words $x_i$ are nonempty. Then an ordered factorization is built in the following way. The set of indices $I_F$ is

$$I_F = \{(w, x_1, \ldots, x_k, i) : (w, x_1, \ldots, x_k) \in F \text{ and } 1 \leq i \leq k\}$$

with the ordering

$$(w, x_1, \ldots, x_k, 1) \geq (w, x_1, \ldots, x_k, 2) \geq \ldots \geq (w, x_1, \ldots, x_k, k).$$
The relationship with the other elements of \( I_F \) is arbitrary. Then we set

\[ X_{\{w, x_1, \ldots, x_k, i\}} = \{x_i\}. \]

Clearly, the constructed ordered factorization defines exactly the factorization \( F \).

The definition of an ordered factorization is too general for algorithmic purposes. A reasonable restriction assumes the finiteness of the set of indices \( I \). Such a factorization can be described in terms of our \( \mathcal{F} \)-factorization.

**Lemma 2.6.** Each finite ordered factorization is an \( \mathcal{F} \)-factorization.

**Proof.** Let \((X_i)_{i \in I}\) be an ordered factorization with \( I = \{1..k\} \). Then we take \( L_i = X_i \) for \( i \in I \) and \( L = \{i_0 \ldots i_k : i_j \in I \text{ and } i_0 \geq i_1 \geq \cdots \geq i_k\} \).

The above requirement of the finiteness possesses some limitations. Indeed, consider for instance the Lyndon factorization: each word can be expressed uniquely in the form \( x_1x_2 \ldots x_k \) where all \( x_i \)'s are Lyndon words and \( x_i \) is lexicographically not smaller than \( x_j \) for \( i < j \), see [9].

**Lemma 2.7.** The Lyndon factorization cannot be expressed as an \( \mathcal{F} \)-factorization.

**Proof.** Suppose the contrary that the Lyndon factorization can be expressed as an \( \mathcal{F} \)-factorization \( \mathcal{F} = (L, L_1, \ldots, L_k) \). Since the set of factors in Lyndon factorizations is the set of Lyndon words \( L_\text{yn} \), we have

\[ L_\text{yn} = L_1 \cup L_2 \ldots L_k. \]

Take any word \( w \) containing \( k+1 \) distinct factors in the Lyndon factorization. Then two of them, say \( x_1, x_2 \) with \( x_1 \) lexicographically greater than \( x_2 \), belong to the same set \( L_i \).

By switching these we obtain a proper \( \mathcal{F} \)-factorization which is not Lyndon.

3. Properties of factorizations

We consider the following three properties of \( \mathcal{F} \)-factorizations:

- **Completeness:** each word over \( \Sigma \) has an \( \mathcal{F} \)-factorization.
- **Uniqueness:** each word has at most one \( \mathcal{F} \)-factorization.
- **Synchronization:** The following condition is satisfied for some nonnegative integer parameters \( l \) and \( r \). Let \( y_1, \ldots, y_k \) and \( x_1, \ldots, x_l \) be \( \mathcal{F} \)-factorizations of words \( y \) and \( x \), respectively. There is a pair of numbers \( l', r' \) satisfying \( l' \leq l \) and \( r' \leq r \) such that the following conditions hold. Denote \( u = y_1 \ldots y_{l'} \) and \( v = y_{k-r'+1} \ldots y_k \). Then, if \( y \) occurs in \( x \) starting at position \( i \), the following conditions are satisfied:
  - positions \( i + |u| \) and \( i + |y| - |v| \) in \( x \) are starting positions of factors, say \( x_p \) and \( x_q \), respectively,
Fig. 1. An illustration of the (2,3)-synchronization property and the structure of the words $y$ and $x$ from the point of view of the automaton $A$ from the proof of Theorem 4.2.

- the sequences of factors $x_{p}, \ldots, x_{q-1}$ and $y_{r'+1}, \ldots, y_{k-r'}$ are identical, i.e.
  $$q - p = k - r' - l'$$ and $x_{p+j} = y_{r'+1+j}$ for $0 \leq j \leq k - r' - l' - 1$,
- the occurrence of $u$ at position $i$ in $x$ covers at most $l - 1$ factors of $x$, i.e. $u$ is a suffix of $x_{\max\{p-1,1\}} \ldots x_{p-1}$,
- the occurrence of $v$ at position $i + \vert y \vert - \vert v \vert$ in $x$ covers at most $r - 1$ factors of $x$, i.e. $u$ is a prefix of $x_{q} \ldots x_{\min\{q+r-1,s\}}$.

Our third condition looks a bit artificial, but, as we shall see, it has a natural counterpart in the case of codes, and it was important in factorizing solutions of word equations in [7]. Intuitively, it says that an $F$-factorization of a subword $y$ of $x$ is the same (with the exception of the first $l$ and the last $r$ factors) as the factorization which is obtained from the $F$-factorization of $x$, see Fig. 1.

Clearly, trivial and block factorizations satisfy all the above conditions. Moreover, the trivial factorization is the only one which satisfies the synchronization condition with parameters $l = r = 0$. Block factorizations satisfy this condition with parameters $l = r = 1$.

An $F$-factorization satisfies the completeness iff $\Sigma \subseteq F$, and it satisfies the uniqueness iff $F$ is a code. The situation with the synchronization is more complicated. In case of codes an $F$-factorization has a synchronization property iff $F$ is a synchronous code, see [8].

**Lemma 3.1.** Both the $F_Q$-factorization and the $F'_Q$-factorization satisfy three properties listed above.

**Proof.** That these factorizations are complete is trivial, and the fact that they are unique follows from a well-known lemma in combinatorics of words: a primitive word $Q$ can be a factor of $QQ$ only in a trivial way.

The most nontrivial point is that these factorizations satisfy the synchronization condition. They satisfy it with parameters $l = r = 2$. Indeed, the factorizations $F_Q, F'_Q$ of a word $w$ are determined by the occurrences of $Q^2$ in $w$. Hence, a factorization of a subword $y$ of $x$ is different from that which is obtained from cutting off $y$ from the
factorization of x, if y starts or ends inside some occurrence of $Q^2$ in x. Therefore the difference is in at most first or last two factors of y. □

Note that the uniqueness of $F_Q$ (or $F_Q'$) factorizations need not hold if $Q$ is not primitive. On the other hand, if $Q$ is not only primitive but also unbordered, i.e. does not contain a common nonempty word as a prefix and as a suffix, then in the definition of the $F_Q$-factorization the second, the third and the fourth points can be removed and in the first point $Q^2$ can be replaced by $Q$, and still we would obtain an $F$-factorization which is complete, unique, and synchronizing.

4. Regular $F$-factorizations

We show that most properties related to regular $F$-factorizations have polynomial time algorithms, the exception being the completeness property.

**Theorem 4.1.** Testing the completeness of regular $F$-factorizations is P-SPACE complete even if the input is specified by deterministic finite automata.

**Proof.** The problem is in P-SPACE since given $L$ and $L_i$'s it is possible to construct a regular expression for words which are factorizable by changing the index $i$ in the expression for $L$ by the expression for $L_i$. Now the problem of the completeness is to check whether the resulting expression corresponds to $\Sigma^*$. This problem is P-SPACE complete, see [6].

The P-SPACE-hardness of the problem for a deterministic representation of a regular factorization $F$ is proved by reduction to the problem of the equivalence of a regular expression and $\Sigma^*$. An NFA representing the input regular expression is transformed into deterministic automaton for $L$ by relabelling its transitions by different letters. Then each such a letter $i$ is associated with a language $L_i = \{a\}$ in such a way that if two transitions labeled now by $i$ and $j$ were labeled (before relabelling) by the same letter then we set $L_i = L_j$. □

For a factorization $F$ and a property $\mathcal{P}$ of factorizations define the language BadWords$(\mathcal{P}, F)$ as the set of all words for which there exists an $F$-factorization not satisfying $\mathcal{P}$. We note here that a factorization $F$ has the property $\mathcal{P}$ iff the language BadWords$(\mathcal{P}, F)$ is empty. If this language is regular, and the construction of the corresponding NFA can be done in polynomial time, then a polynomial time test algorithm to check the emptiness of regular languages can be used to derive a polynomial time algorithm to decide whether $F$ possesses the property $\mathcal{P}$.

We consider now two basic properties $\mathcal{P}$ for which BadWords$(\mathcal{P}, F)$ is regular.

**Theorem 4.2.** Let $l$ and $r$ be integers and $F$ a regular $F$-factorization. There is a polynomial time algorithm to test whether $F$ possesses the $(l,r)$-synchronization property.
Proof. We construct an NFA accepting \( \text{BadWords}(\mathcal{P}, \mathcal{F}) \), where \( \mathcal{P} \) is the property of being \((l, r)\)-synchronized. The automaton \( A \) accepts a word \( x \) iff there is a subword \( y \) of \( x \) such that the pair \((y, x)\) contradicts the \((l, r)\)-synchronization property. \( A \) is a parallel composition of several automata and we describe its construction only informally by referring to Fig. 1.

Assume \( \mathcal{F} = (L, L_1, \ldots, L_k) \) and let NFA's \( B, A_1, \ldots, A_k \) accept languages \( L, L_1, \ldots, L_k \), respectively.

\( A \) reads an input word \( x \) symbol by symbol each time guessing decomposition into factors. Whenever a starting position of a factor belonging to \( L_i \) is guessed the automaton \( A_i \) is activated. When \( A_i \) arrives in an accepting state this means that one full factor is terminated. The consecutive indices of terminating factors are passed to \( B \) which verifies whether the sequence of indices of factors is in \( L \).

The automaton guesses a starting position \( i \) and the last position \( j \) of an occurrence of a word \( y \) in \( x \), which is also guessed symbol by symbol.

The numbers \( l, r \) are fixed so that they can be kept in the finite memory of the automaton, two counters are used to count from 1 to \( \max(l, r) \). \( A \) checks an agreement of factors in the synchronized part, see Fig. 1. Since the size of \( A \) is polynomial in the size of \( \mathcal{F} \) and \((l, r)\), it can be checked in polynomial time if \( A \) accepts any word. \( \square \)

**Theorem 4.3.** There is a polynomial time algorithm to test the uniqueness property for regular factorizations.

Proof. We construct a nondeterministic finite automaton accepting the language \( \text{BadWords}(\mathcal{P}, \mathcal{F}) \), where \( \mathcal{P} \) is in this case the property that the input word \( y \) has two different \( \mathcal{F} \)-factorizations. Similarly as in Theorem 4.2 we construct a corresponding automaton \( A \). Now it accepts an input word \( y \) iff there are two \( \mathcal{F} \)-factorizations of \( y \). The automaton reads the word \( y \) from left to right and guesses starting and ending positions of two factorizations. Each guessed factor is checked by an automaton \( A_i \). If two different \( \mathcal{F} \)-factorizations are found then \( y \) is accepted. So the result follows since \( A \) is polynomial in size of \( \mathcal{F} \), and the emptiness of NFA's can be solved in polynomial time. \( \square \)

A labeled factorization is a factorization of a given word together with a sequence of indices of languages \( L_i \) which correspond to the factorization. A given factorization may correspond to two different labeled factorizations.

**Theorem 4.4.** Assume we are given a regular factorization \( \mathcal{F} = (L, L_1, \ldots, L_k) \) of size \( n \) and an input word \( w \) of size \( m \).

1. Then we can find an \( \mathcal{F} \)-factorization of \( w \) in polynomial time with respect to \( n+m \), or find out that there is no \( \mathcal{F} \)-factorization of \( w \).
2. We can compute the total number \( N' \) of labeled \( \mathcal{F} \)-factorizations of \( w \) in polynomial time if a deterministic automaton \( B \) accepting \( L \) is given. Moreover, if the
languages \(L_1, \ldots, L_k\) are disjoint, then the total number \(N\) of \(F\)-factorizations of \(w\) can be computed in polynomial time.

**Proof.** Denote by \(w[p..q]\) the subword of \(w\) starting at position \(p\) and ending at position \(q\). For each subword \(w[p..q]\) and each \(1 \leq i \leq k\) we check whether \(w[p..q] \in L_i\). This can be done in \(O(m^2n)\) time.

We construct an acyclic labeled multigraph \(G\) the nodes of which are positions in \(w\) and the number 0. For each \(t, s, i\) we create an edge \((t, s)\) labeled by \(i\) iff \(w[t..s] \in L_i\).

Let \(B\) be an automaton accepting \(L\). We construct an acyclic multigraph \(G'\) the nodes of which are pairs \((t, q)\), where \(t\) is a node of \(G\) and \(q\) is a state of \(B\). There is an edge labeled by \(i\) from \((t, q)\) to \((t', q')\) iff \(q' \in \delta_B(q, i)\) and there is an edge labeled by \(i\) from \(t\) to \(t'\) in \(G\). The source node of \(G'\) is \((0, q_B^0)\) and the sinks of \(G'\) are the pairs \((m, q_a)\), where \(q_a\) is any accepting state of \(B\).

Now part (1) reduces to testing whether there is a path from a source of \(G'\) to a sink node of \(G'\).

If \(B\) is deterministic then the number of all paths from a source of \(G'\) to a sink node of \(G'\) equals to the number \(N'\) requested in part (2). The number of all paths in an acyclic multigraph from a given node to another can be easily computed in polynomial time by processing the nodes in the reversed topological order.

Finally, if additionally the languages \(L_1, \ldots, L_k\) are disjoint then \(G\) is a graph without multiple edges, and the number \(N\) of different \(F\)-factorizations is similarly computed as the number of different paths in the corresponding multigraph \(G'\). \(\square\)

Note that if \(B\) above is a nondeterministic acyclic automaton then it accepts a finite language \(L\), however the (deterministic) polynomial time algorithm computing the cardinality of \(L\) cannot be concluded – at least by using the above reasoning.

For factorizations \(F_Q\) and \(F'_Q\) we have even a better result.

**Lemma 4.5.** The \(F_Q\)- and \(F'_Q\)-factorizations of a word \(w\) can be computed in time \(O(|w|)\).

**Proof.** The occurrences of the word \(Q^2\) inside \(w\) determine how to divide the word \(w\) into factors. The searching for \(Q^2\) can be done by any of linear time pattern-matching algorithms, see [4]. \(\square\)

5. **Context-free \(F\)-factorizations**

The complexity of problems considered above changes drastically if we replace regular \(F\)-factorizations by context-free ones. In the latter case testing of the uniqueness, as well as the completeness become undecidable.

**Theorem 5.1.** The completeness problem for context-free \(F\)-factorizations is undecidable.
Proof. Let \( \mathcal{F} = (L, L_1, \ldots, L_k) \) where \( L \) and \( L_i \)'s are given by context-free grammars \( G \) and \( G_i \)'s, respectively. Now, we construct a grammar \( G' \) by replacing the terminal \( i \) of \( G \) by the starting symbol of \( G_i \). The language generated by \( G' \) contains exactly those words which are \( \mathcal{F} \)-factorizable. Now, if the language \( L_i \) consists of single one-letter word then the resulting grammar may be arbitrary. Hence, the problem of the theorem is equivalent to whether an arbitrary context-free grammar produces the language \( \Sigma^* \), which is undecidable, see [6].

Theorem 5.2. Testing the uniqueness property of context-free \( \mathcal{F} \)-factorizations is undecidable.

Proof. The emptiness of the intersection of two context-free languages is undecidable, see [6]. Let \( L_1, L_2 \) be arbitrary context-free languages over the alphabet \( \Sigma = \{a, b\} \). Consider an \( \mathcal{F} \)-factorization \( (L, L_1, L_2) \) for \( L = \{1, 2\} \). Now, the \( \mathcal{F} \)-factorization is unique iff \( L_1 \cap L_2 = \emptyset \). This completes the proof.

Note here that in the above theorems the finiteness of \( L \) does not change the complexity of the problems. Indeed, for \( L = \{1\} \) and \( L_1 \) equal to an arbitrary context-free language the completeness remains undecidable.

Theorem 5.3. Assume we are given a context-free factorization \( \mathcal{F} = (L, L_1, \ldots, L_k) \) of size \( n \) and an input word \( w \) of size \( m \). Then we can find an \( \mathcal{F} \)-factorization of the input word in polynomial time with respect to \( n+m \), or find out that there is no \( \mathcal{F} \)-factorization.

Proof. We use a modification of the classical algorithm by Cocke–Younger–Kasami for the membership problem for context-free languages (the CYK algorithm for short), see [6]. The algorithm uses the dynamic programming technique to compute, for each subword of the input word, the set of nonterminals from which they are derivable. We use the CYK algorithm \( k \) times to compute, for each language \( L_r \) all subwords \( w[i..j] \) of \( y \) such that \( w[i..j] \in L_r \). Next, using the same technique we use the above information to compute, for each subword of \( y \), the set of nonterminals of \( L \) which derive \( \mathcal{F} \)-factorizations of \( y \).

Observe here that we cannot have much better algorithm in the above since the problem is more difficult than the membership problem for context-free languages. Indeed, if \( L_i \) are one-letter languages then the problem is to find a derivation of an input word \( y \) in an arbitrary context-free grammar.

6. Open problems

We conclude by posing a few open problems:

- Find efficient algorithms for the polynomial time solvable problems we discussed above.
• Given a word, can its minimal and maximal regular $F$-factorizations, in the sense of the length of the sequence of indices, be found in polynomial time?

• Could the better algorithms be designed for our problems if in regular $F$-factorizations only finite languages are considered? Is the completeness and the uniqueness undecidable if context-free $F$-factorizations are given by deterministic automata or by linear context-free grammars?

• What is the complexity of the problem of determining whether a regular $F$-factorization possesses synchronization property if the parameters $l$, $r$ are not given? What about the complexity of the problem for context-free $F$-factorizations?

Acknowledgements

We would like to thank an anonymous referee whose detailed comments helped us to improve the presentation of our paper.

References


