Testing equivalence of morphisms on context-free languages *

Wojciech Plandowski [1] **

Instytut Informatyki UW, 02-097 Banacha 2, Warszawa, Poland

Abstract. We present a polynomial time algorithm for testing if two morphisms are equal on every word of a context-free language. The input to the algorithm are a context-free grammar with constant size productions and two morphisms. The best previously known algorithm had exponential time complexity. Our algorithm can be also used to test in polynomial tiime whether or not n first elements of two sequences of words defined by recurrence formulae are the same. In particular, if the well known $2n$ conjecture for D0L sequences holds, the algorithm can test in polynomial time equivalence of two D0L sequences.
Additionally, we extend the result from [5] by proving the existence of polynomial size test sets for context-free languages not only in free monoids but in free groups as well. The main points of our proof are the same as in [6, 5]. The main change is a new short proof of the main lemma. The previous proof took 10 pages. It was complicated since it considered many cases and used advanced properties of periodicity of words. Our proof takes only 2 pages. The simplification is a consequence of embedding a free monoid into a free group.

1 Introduction

The problem of equivalence of morphisms on context-free languages (cfl, for short) consists in checking for given two morphisms f, g and a context-free grammar G (cfg, for short) if they coincide on $L(G)$ (i.e. if $f(w) = g(w)$ for $w \in L(G)$). The idea to solve the problem is connected to the notion of a test set. A set of words T is a test set for a language L iff T is a subset of L and for every two morphisms f, g if they coincide on T then they coincide on L. Hence, it is enough to compare morphisms on words from a test set to decide if they coincide on a whole language. The existence of finite test sets for cfls was proved in [2]. The number of words in the test set could be double exponential. This result was improved to single exponential [4] and later to polynomial, see [6]. The lengths of words in the test set however could be exponential. Moreover, the length of the shortest word in a cfl defined by a grammar with n productions can be exponential, see Example 1. Thus, for some cfls it is impossible to list in polynomial time all words from any test set. To avoid this problem we encode

* Supported by the grant EC Cooperation Action IC 1000 Algorithms for Future Technologies ALTEC.

** E-mail: wojtekpl@mimuw.edu.pl

every word from the test set by a small cfg. Then for each word w from the test set we find codes for $f(w)$, $g(w)$. Finally, we present a polynomial time algorithm to test if two codes represent the same word. The algorithm uses similar ideas as the Makanin algorithm (which is of exponential complexity) for testing whether or not a word equation has a solution, see [9] and later improvements [3, 7, 10, 11].

We generalize the definition of a test set to all monoids. A subset T of a language L is a test set for L in a monoid M iff for each two morphisms f, $g : \Sigma^* \to M$ if they coincide on T then they coincide on L. We prove the existence of polynomial size test sets for cfls in free groups. At one stroke we extend the result from [5] and obtain a shorter proof! As an example how it is possible we present two proofs of the fact which is used in the proof of Lemma 2. The proof in a free monoid considers two cases while the proof for groups uses inverse elements instead.

Lemma 1. *$X = \{uv,\ uw,\ xv\}$ is a test set for $Y = \{uv,\ uw,\ xv,\ xw\}$ in every group.*

Proof. (in a free monoid)
Let f, g be two morphisms. It is enough to prove that if $f(z) = g(z)$ for $z \in X$ then $f(xw) = g(xw)$. Denote $s' = f(s)$ and $s'' = g(s)$ for any word s. Since $u'v' = u''v''$ two cases are to be considered:
Case 1: u' is a prefix of u''.
There is a word σ such that $u'' = u'\sigma$. Using it in equations $u'v' = u''v''$ and $u'w' = u''w''$ and simplifying them we have $v' = \sigma v''$ and $w' = \sigma w''$, respectively. Since $x'v' = x''v''$ we obtain $x'' = x'\sigma$ and finally $x''w'' = x'\sigma w'' = x'w'$.
Case 2: u'' is a prefix of u'.
The proof in that case is symmetric to the one in the previous case.

Proof. (in a group)
Let f, g be two morphisms. It is enough to prove that if $f(z) = g(z)$ for $z \in X$ then $f(xw) = g(xw)$. Denote $s' = f(s)$ and $s'' = g(s)$ for any word s. We have

$$x'w' = x'v'(v')^{-1}(u')^{-1}u'w' = x'v'(u'v')^{-1}u'w' = x''v''(u''v'')^{-1}u''w'' = x''w''$$

and we are done.

The fact above is an extension of the main fact in [4] and allows to prove using the same construction as in [4] that in every group every regular language has a linear size test set.

2 Polynomial size test sets for context-free languages

Recall, that a cfg is in *Chomsky normal form* if each production of the grammar is either in form $A \to BC$ or in form $A \to w$ where A, B, C are nonterminal symbols and w is a terminal symbol. Since every cfg whose productions are of constant size can be transformed in polynomial time to a cfg in Chomsky normal form we restrict our considerations to cfgs which are in that form. A

linear context-free grammar is a cfg whose productions are of the form $A \to uBv$ or $A \to u$ where u, v are terminal words and A, B nonterminal symbols. For every nonterminal A let w_A be any shortest word derivable from A. For a cfg G in Chomsky normal form we define a linear cfg $lin(G)$ by replacing every production of form $A \to BC$ by three productions $A \to w_B C$, $A \to B w_C$, $A \to w_B w_C$.

Lemma 2. *Let G be a cfg in Chomsky normal form. Then $L(lin(G))$ is a test set for $L(G)$ in every group.*

Proof. The proof is the same as the proof of Lemma 2 in [5], see also the proof of Theorem 1 in [6]

As we shall see in the next lemma to check if in a given group the size of a minimal test set can be bounded by a polynomial it is enough to consider test sets for finite languages L_k which are defined by the following linear cfg G_k:

$$A_i \to a_i A_{i-1} \bar{a}_i \mid b_i A_{i-1} \bar{b}_i \qquad \text{for } 1 < i \leq k$$
$$A_1 \to a_1 \mid b_1$$

where nonterminals are in capitals, terminal symbols in lower case letters and A_k is the starting symbol. The language L_k consists of 2^k words and it is defined over an alphabet consisting of $4k-2$ symbols. Let μ_k be the only word from L_k consisting of letters b_i and $\bar{b}_i$. Define $T_k = L_k - \{\mu_k\}$.

A linear cfg G can be viewed as the digraph $graph(G)$ whose vertices correspond to nonterminals and there is an edge from A to B labeled by a pair of terminal words (u, v) if $A \to uBv$ is a production in G. Additionally we add a terminal node t. There is an edge from A to t labelled (u, v) if $v = \epsilon$ and there is a production $A \to u$ in G. In such a graph every path from the starting nonterminal S to t corresponds to a derivation in G of a word from $L(G)$. On the other hand each derivation of a word from $L(G)$ has its corresponding path in $graph(G)$ from S to t. A word w whose derivation corresponds to a path $\lambda = (S, v_1), \ldots, (v_k, t)$ can be obtained in the following way. If $(w_0, \bar{w}_0), \ldots, (w_k, \bar{w}_k)$ are labels corresponding to consecutive edges of λ then $w = w_0 \ldots w_k \bar{w}_k \ldots \bar{w}_0$.

For each node v of $graph(G)$ we build a tree $tree(v)$ rooted at v containing all vertices reachable from v in $graph(G)$. With each sequence of not necessarily adjacent edges $(u_1, v_1), \ldots, (u_k, v_k)$ we associate a path which starts at S goes in $tree(S)$ to u_1 then it traverses (u_1, v_1) and then goes in $tree(v_1)$ to a vertex u_2 runs through (u_2, v_2) etc. Finally, it goes via (u_k, v_k) and $tree(v_k)$ to t. Observe, that for some sequences of edges there are no paths which satisfy the conditions above and since there is exactly one path in a tree which starts at the root and ends with a given vertex of the tree the path if exists is unique. Let $F_k^{tree}(G)$ be a set of words corresponding to paths associated with a sequence of at most k edges. The index $tree$ in $F_k^{tree}(G)$ means that the contents of this set depends on the function $tree$. The number of words in $F_k^{tree}(G)$ does not exceed the number of sequences of at most k edges. Since the number of edges of $graph(G)$ is equal to the number of productions in G, the size of $F_k^{tree}(G)$ does not exceed

$\sum_{i=0}^{k} n^i \leq (k+1)n^k$ where n is the number of productions in G. Suppose the sizes of productions from G are $O(1)$. Since the length of a path in a tree embedded in $graph(G)$ does not exceed n the length of a path associated with a sequence of k edges does not exceed kn and the words in $F_k(G)$ are not longer than $O(kn)$.

Lemma 3. *Let G be a linear cfg and Gr be a group. If T_k is a test set for L_k in Gr then $F^{tree}_{2k-2}(G)$ is a test set for $L(G)$ in Gr.*

Proof. The proof is a straightforward generalization of the proof of Lemma 3 from [5], see also the proof of Lemma 2 in [6].

Theorem 4. *Let Gr be a group. There is a polynomial upper bound in Gr for the size of minimal test sets for context-free languages which are defined by context-free grammars with constant size productions iff there is $k > 0$ such that T_k is a test set for L_k in Gr.*

Proof. Let G be a cfg in Chomsky normal form with n productions. If there exists $k > 0$ such that T_k is a test set for L_k then, by Lemma 3 and Lemma 2, $F^{tree}_{2k-2}(lin(G))$ is a test set for $L(G)$. Since $F^{tree}_{2k-2}(lin(G))$ contains at most $(2k-1)(3n)^{2k-2}$ words the polynomial upper bound exists. Suppose, there does not exist $k > 0$ such that T_k is a test set for L_k. Since the number of words in L_k is 2^k and it is defined by a grammar G_k which can be easily transformed to a grammar in Chomsky normal form with $O(k)$ productions the result follows.

By Theorem 4, to prove in a group the existence of polynomial size test set for cfl it is enough to find k such that T_k is a test set for L_k in the group. The next lemma proves that for $k = 4$ it is true in free groups.

Denote $w^a = a^{-1}wa$. Note, that $(w^a)^b = w^{ab}$

Lemma 5. *Consider morphisms f, g. Denote $s' = f(s)$ and $s'' = g(s)$ for each element s of a free group. Let τ, ρ, ω, σ, $\overline{\sigma}$, a, b, c, d be elements of the free group. Then*

1. *If $\omega\tau = \tau\omega$ and $\rho\tau = \tau\rho$ and $\tau \neq 1$ then $\omega\rho = \rho\omega$.*
2. $$\begin{cases} \sigma^a\tau^c = \tau^c\sigma^a \\ \sigma^b\tau^c = \tau^c\sigma^b \\ \sigma^a\tau^d = \tau^d\sigma^a \end{cases} \Longrightarrow \sigma^b\tau^d = \tau^d\sigma^b \ ,$$
3. *If $\sigma x' = x'\overline{\sigma}$ for $x \in T_3$ then $\sigma\mu'_3 = \mu'_3\overline{\sigma}$,*
4. *If $x' = x''$ for $x \in T_4$ then $\mu'_4 = \mu''_4$.*
5. *T_4 is a test set for L_4 in a free group.*

Proof.

1. The proof of this point is a consequence of the fact that every subgroup of a free group is free. We omit details.

2. It is a consequence of point 1. We may assume that $\sigma \neq 1$ and $\tau \neq 1$. We apply point 1 to the first two equations to obtain $\sigma^a \sigma^b = \sigma^b \sigma^a$. Now, we apply point 1 again to obtained equation and the third equation.

3. We have $\overline{\sigma} = (x')^{-1} \sigma x' = \sigma^{x'}$ for $x \in T_3$. Hence $\sigma^{x'} = \sigma^{y'}$ for $x, y \in T_3$ and therefore $\sigma^{x' y'^{-1}} = \sigma$ for $x, y \in T_3$. For $x = a_3 a_2 a_1 \bar{a}_2 \bar{a}_3$, $y = a_3 a_2 b_1 \bar{a}_2 \bar{a}_3$ we have $\sigma^{a'_3 a'_2 a'_1 b_1^{-1} a'^{-1}_2 a'^{-1}_3} = \sigma$ (α). Similarly, we obtain $\sigma^{a'_3 b'_2 a'_1 b_1^{-1} b'^{-1}_2 a'^{-1}_3} = \sigma$ (β) and $\sigma^{b'_3 a'_2 a'_1 b_1^{-1} a'^{-1}_2 b'^{-1}_3} = \sigma$ (γ). Now, we transform the equation (α). Let $\rho = a'_1 b_1^{-1}$. We have $\sigma^{a'_3 \rho^{a'^{-1}_2} a'^{-1}_3} = \sigma$. Thus $\sigma^{a'_3 \rho^{a'^{-1}_2}} = \sigma^{a'_3}$ and therefore $(\sigma^{a'_3})^{\rho^{a'^{-1}_2}} = \sigma^{a'_3}$. Finally, we obtain $\sigma^{a'_3 \rho^{a'^{-1}_2}} = \rho^{a'^{-1}_2} \sigma^{a'_3}$. Similarly, we transform equations (β), (γ) to have $\sigma^{a'_3 \rho^{b'^{-1}_2}} = \rho^{b'^{-1}_2} \sigma^{a'_3}$, $\sigma^{b'_3 \rho^{a'^{-1}_2}} = \rho^{a'^{-1}_2} \sigma^{b'_3}$, respectively. Now, we apply point 2 to obtain $\sigma^{b'_3 \rho^{b'^{-1}_2}} = \rho^{b'^{-1}_2} \sigma^{b'_3}$. By transforming the last equation in the reverse order than previously we obtain $\sigma^{b'_3 b'_2 a'_1 b_1^{-1} b'^{-1}_2 b'^{-1}_3} = \sigma$ which is equivalent to $\sigma^{\mu'_3} = \sigma^{\nu'}$ where $\nu = b_3 b_2 a_1 \bar{b}_2 \bar{b}_3$. Since $\nu \in T_3$ we have $\sigma^{\nu'} = \overline{\sigma}$ and finally $\sigma^{\mu'_3} = \overline{\sigma}$ which is equivalent to $\mu'_3 \overline{\sigma} = \sigma \mu'_3$.

4. The system of equations $x' = x''$ for $x \in T_4$ can be written in form

$$\begin{aligned} a'_4 x' \bar{a}'_4 &= a''_4 x'' \bar{a}''_4 \qquad \text{for } x \in L_3 \\ b'_4 x' \bar{b}'_4 &= b''_4 x'' \bar{b}''_4 \qquad \text{for } x \in T_3. \end{aligned}$$

Hence for $x \in T_3$ we have
$a'_4 x' \bar{a}'_4 = a''_4 x'' \bar{a}''_4 = a''_4 b''^{-1}_4 (b''_4 x'' \bar{b}''_4) \bar{b}''^{-1}_4 \bar{a}''_4 = a''_4 b''^{-1}_4 b'_4 x' \bar{b}'_4 \bar{b}''^{-1}_4 \bar{a}''_4$
and therefore $(b'^{-1}_4 b''_4 a''^{-1}_4 a'_4) x' = x' (\bar{b}'_4 \bar{b}''^{-1}_4 \bar{a}''_4 \bar{a}'^{-1}_4)$. Now, we apply point 3 and obtain $(b'^{-1}_4 b''_4 a''^{-1}_4 a'_4) \mu'_3 = \mu'_3 (\bar{b}'_4 \bar{b}''^{-1}_4 \bar{a}''_4 \bar{a}'^{-1}_4)$.
Hence, we have $b''_4 a''^{-1}_4 (a'_4 \mu'_3 \bar{a}'_4) \bar{a}''^{-1}_4 = b'_4 \mu'_3 \bar{b}'_4 \bar{b}''^{-1}_4$. Since $a'_4 \mu'_3 \bar{a}'_4 = a''_4 \mu''_3 \bar{a}''_4$ we have $b'_4 \mu'_3 \bar{b}'_4 = b''_4 \mu''_3 \bar{b}''_4$ and finally $\mu'_4 = \mu''_4$.

5. This point is a reformulation of point 4.

We say, that a grammar G *defines a word* w if it is a cfg in Chomsky normal form without useless nonterminals such that each nonterminal is on the left hand side of exactly one production and $L(G) = \{w\}$. In such a grammar the word w has exactly one derivation.

Example 1. Some grammars define a word of exponential size with respect to the number of their productions, e.g. the grammar with productions $A_i \to A_{i-1} A_{i-1}$ for $1 \leq i \leq k$, $A_0 \to a$ and the starting symbol A_k defines the word a^{2^k}. We treat a grammar defining w as a compressed representation of w.

Theorem 6. *In a free group every cfl generated by a grammar in Chomsky normal form with n productions has a test set of size at most $7(3n)^6$. Grammars defining every word from the test set can be found in polynomial time.*

Proof. Let G be a cfg in Chomsky normal form with n productions. By Lemma 2, Lemma 3 and Lemma 5 $F_6^{tree}(lin(G))$ is a test set for $L(G)$. Since $lin(G)$ has at most $3n$ productions $F_6^{tree}(lin(G))$ has at most $7(3n)^6$ words and the first part of the theorem is proved.

Since the shortest words derivable from nonterminals in grammar G can be of exponential size the productions of $lin(G)$ can be of exponential size. Fortunately, for each nonterminal A there is a shortest word derivable from A which is defined by a subgrammar of G. Take such a word as a w_A. Now, we treat all w_A as terminal symbols in the grammar $lin(G)$ remembering that those symbols represent words which are defined by grammars. On the basis of the graph for $lin(G)$ we find $F_6(lin(G))$. The words are of lengths $O(n)$ and they can be found in polynomial time. They consist of terminal symbols of $lin(G)$ which correspond to words defined by grammars. Now, for each word from the test set we can find a grammar defining this word.

3 A polynomial time algorithm for testing if two morphisms coincide on a context-free language

From now on we deal with a free monoid. Let f be a morphism and let G be a grammar defining w. A grammar defining the word $f(w)$ is easily derived from G by replacing every production of form $A \to u$ where u is a terminal symbol by $A \to f(u)$. By Theorem 6 to find a polynomial time algorithm for morphism equivalence problem for a cfl it is enough to find a polynomial time algorithm for testing if two grammars define the same word. We describe a polynomial time algorithm for a more general problem.

We say, that a grammar G *defines a set of words* S if the grammar is context-free in Chomsky normal form (there can be useless nonterminals), each nonterminal is on left hand side of exactly one production, every nonterminal generates exactly one terminal word and for each $w \in S$ there is a nonterminal A in G such that $A \to^* w$. In particular, if G defines a word then it defines a set of words derivable from nonterminals of G. Denote by w_A the terminal word derivable from A in G if A is a nonterminal and A if A is a terminal symbol.

Example 2. For fixed k consider a grammar G containing the following set of productions:

$$\begin{array}{ll} A_i \to A_{i-1}A_{i-1} \text{ for } 1 \le i \le k & A' \to A_0 A'_k \\ A_0 \to a & A'_i \to A'_{i-1}A_{i-1} \text{ for } 1 \le i \le k \\ & A'_1 \to a \end{array}$$

Since $w_{A_i} = a^{2^i}$ for $0 \le i \le k$, $w_{A'_i} = a^{2^i-1}$ for $1 \le i \le k$, $w_{A'} = a^{2^k}$ the grammar defines the set of words

$$\{a^{2^i} : 0 \le i \le k\} \cup \{a^{2^i-1} : 1 \le i \le k\}.$$

We describe a polynomial time algorithm for the following problem:

The Problem

Let G be a grammar which defines a set of words and S be a set of pairs of nonteminals from G. Check if $w_A = w_B$ for every pair (A, B) from S.

Let G be a grammar with n productions which defines a set of words W. Since the shortest words derivable from a nonterminal of a grammar in Chomsky

normal form with n productions are of size not exceeding 2^n the lengths of words from W are not longer than 2^n. Such numbers can be stored in n bits. Standard algorithms for basic operations (comparing, addition, substraction, division, multiplication) on such numbers work in polynomial time with respect to n. This allows to compute the length of every word from W in polynomial time.

The key point of our algorithm is to consider relations between words w_A. The relations are stored as sets of triples $(A, B.i)$ where A, B are nonterminal or terminal symbols and i is a nonnegative integer. We divide the triples into two groups which we call *suffix* and *subword* triples. Let $|w|$ be the length of the word w. A triple (A, B, i) is a suffix triple iff $i + |w_B| \geq |w_A|$ and it is a subword triple iff $i + |w_B| < |w_A|$ and $i \geq 1$. The basic relation in our considerations is $SUFSUB(G)$. Intuitively, a triple (A, B, i) is in $SUFSUB(G)$ iff starting at position $i + 1$ in word w_A consecutive symbols of w_A are the same as in w_B starting from the beginning. More precisely:

$(A, B, i) \in SUFSUB(G)$ iff
(A, B, i) is a suffix triple and $w_A[i + 1 \ldots |w_A|] = w_B[1 \ldots |w_A| - i]$ or
(A, B, i) is a subword triple and $w_A[i + 1 \ldots i + |w_B|] = w_B$

Now, The Problem can be refomulated as follows:
Given a grammar G defining a set of words and a set S of pairs of nonterminals from G. For each pair (A, B) from S check if $|w_A| = |w_B|$ and $(A, B, 0) \in SUFSUB(G)$.

Let *rel* be a set of triples. We define an operation $Split(A, rel)$ which eliminates occurrences of a nonterminal A from triples of *rel* and does not lose the information from *rel*. Assume, $A \rightarrow EF$ is a production of G. For one triple the operation $Split(A, (B, C, i))$ is defined as follows:

- Case $A \neq B$ and $A \neq C$

$$Split(A, (B, C, i)) = (B, C, i)$$

- Case $A = B$ and $A \neq C$

$$Split(A, (A, C, i)) = \begin{cases} (E, C, i) \cup (C, F, |w_E| - i) \\ \qquad \text{if } |w_E| > i \text{ and } |w_C| + i > |w_E|, \\ (E, C, i) \\ \qquad \text{if } |w_E| > i \text{ and } |w_C| + i \leq |w_E|, \\ (C, F, 0) \\ \qquad \text{if } |w_E| = i \text{ and } |w_C| \leq |w_F|, \\ (F, C, 0) \\ \qquad \text{if } |w_E| = i \text{ and } |w_C| > |w_F|, \\ (F, C, i - |w_E|) \\ \qquad \text{if } |w_E| < i. \end{cases}$$

– Case $A \neq B$ and $A = C$

$$Split(A,(B,A,i)) = \begin{cases} (B,E,i) \\ \qquad \text{if } |w_E| + i \geq |w_B|, \\ (B,E,i) \cup (B,F,|w_E| + i) \\ \qquad \text{if } |w_E| + i < |w_B|. \end{cases}$$

– Case $A = B$ and $A = C$

$$Split(A,(A,A,i)) = \begin{cases} \emptyset \\ \qquad \text{if } i = 0 \\ (E,E,i) \cup (E,F,|w_E| - i) \\ \qquad \text{if } |w_E| > i \geq 1 \text{ and } i \geq |w_F|, \\ (E,E,i) \cup (F,F,i) \cup (E,F,|w_E| - i) \\ \qquad \text{if } |w_E| > i \geq 1 \text{ and } i < |w_F|, \\ (E,F,0) \\ \qquad \text{if } |w_E| = i \text{ and } |w_E| \geq |w_F|, \\ (F,E,0) \cup (F,F,i) \\ \qquad \text{if } |w_E| = i \text{ and } |w_E| < |w_F|, \\ (F,E,i - |w_E|) \\ \qquad \text{if } |w_E| < i \text{ and } i \geq |w_F|, \\ (F,E,i - |w_E|) \cup (F,F,i) \\ \qquad \text{if } |w_E| < i \text{ and } i < |w_F|. \end{cases}$$

For a set of triples *rel* we define $Split(A, rel) = \bigcup_{(B,C,i) \in rel} Split(A,(B,C,i))$. The definition of *Split* looks very complicated, but the idea is to eliminate nonterminal A from triples of *rel* without losing the information about the dependences between words in *rel*. Our next lemma states in formal way that the relation $Split(A, rel)$ is equivalent to *rel*.

Lemma 7. *$rel \subseteq SUFSUB(G)$ iff $Split(A, rel) \subseteq SUFSUB(G)$ for any nonterminal A*

Proof. The proof from left to right is a consequence of the fact that on the basis of the relation *rel* we define *Split* and we do not add dependences which are not derivable from *rel*. The proof from right to left is a consequence of the fact that having *Split* we can restore *rel* since all the information from *rel* were moved to *Split*. The formal proof of the above is simple but tedious. It consists of several cases corresponding to the cases in the definition of *Split*.

Let $\#suffix(rel)$ and $\#subword(rel)$ be the number of suffix and subword triples in *rel*, respectively. Let $\#sufsub(rel) = \#suffix(rel) + \#subword(rel)$. Clearly, $\#sufsub(rel)$ is the number of triples in *rel*. The following lemma will be useful later.

Lemma 8. *Let rel be a set of triples such that for every $(B,C,i) \in rel$ $|w_A| \geq \max\{|w_B|, |w_C|\}$. Then*

$$\#suffix(Split(A, rel)) \leq 3\#sufsub(rel)$$
$$\#subword(Split(A, rel)) \leq \#subword(rel) + 3\#suffix(rel)$$

Proof. The first equality is clear since one triple from *rel* corresponds to at most three triples in $Split(A, rel)$. The second one is a consequence of the fact that every triple from *rel* corresponds to at most one subword triple in *Split*. Look at the definition of *Split*. In most cases triples correspond to one triple. Since (H, H, i) for any H is always a suffix triple it remains to consider the case $A = B$ and $A \neq C$ for $|w_E| > i$ and $|w_C| + i > |w_E|$ and the case $A \neq B$ and $A = C$ for $i + |w_E| < |w_B|$. In the first case (E, C, i) is a suffix triple, in the second one since $|w_A| \geq |w_B|$ $(B, F, |w_E| + i)$ is a suffix triple.

The second operation we define for a set of triples *rel* is the operation $Compact(rel)$. It reduces the number of suffix triples in relation *rel*.

Lemma 9. (periodicity lemma, see [8])
If x, y are periods of a word w and $x + y \leq |w|$ then $gcd(x, y)$ is also a period of w where gcd stands for the greatest common divisor.

Computing the greatest common divisor of two n bit numbers takes polynomial time, see [1].

Define the operation $SimpleCompact(rel)$ in the following way. If there are three suffix triples (A, B, i), (A, B, j), (A, B, k) in *rel* such that $(j-i)+(k-i) \leq |w_A| - i$ and $i < j < k$ then the operation replaces them by two triples (A, B, i), $(A, B, i+gcd(j-i, k-i))$. If there are no such three triples it fails. A $Compact(rel)$ becomes from *rel* by applying *SimpleCompact* to *rel* until it fails.

Lemma 10. *$rel \subseteq SUFSUB(G)$ iff $Compact(rel) \subseteq SUFSUB(G)$*

Proof. It is enough to prove that $rel \subseteq SUFSUB(G)$ iff $SimpleCompact(rel) \subseteq SUFSUB(G)$. If $rel \subseteq SUFSUB(G)$ and *SimpleCompact* does not fail then both $j - i$ and $k - i$ are periods of the word $w = w_B[1 \ldots |w_A| - i]$ and by periodicity lemma $gcd(j - i, k - i)$ is also a period of it. Hence, $(A, B, i + gcd(j - i, k - i))$ is in $SUFSUB(G)$. On the other hand if (A, B, i) and $(A, B, i + gcd(j - i, k - i))$ are in $SUFSUB(G)$ then $gcd(j - i, k - i)$ is a period of w. Then $j - i$, $k - i$ as multiplies of $gcd(j - i, k - i)$ are periods of w and therefore both (A, B, j), (A, B, k) are in $SUFSUB(G)$.

Lemma 11. *If rel is a set of triples then $\#suffix(Compact(rel)) \leq (2n+1)n^2$.*

Proof. Let (A, B, i_1), (A, B, i_2),..., (A, B, i_k) be a sequence of all suffix triples of form (A, B, i) from $Compact(rel)$ sorted on i. Since the operation *SimpleCompact* fails we have $i_{r+2} - i_r + i_{r+1} - i_r > |w_A| - i_r$. Hence, $2i_{r+2} - i_r > |w_A|$ and therefore $\frac{1}{2}(|w_A| - i_r) > (|w_A| - i_{r+2})$. Since $|w_A| - i_1 \leq 2^n$, the sequence of numbers $|w_A| - i_1$, ..., $|w_A| - i_k$ has at most $2n + 1$ elements. Therefore the number of all suffix triples in $Compact(rel)$ does not exceed $(2n + 1) * n^2$.

Algorithm Test;
{ input:
grammar G which defines a set of words
set S of pairs of nonterminals from G

output:
test if for each $(A, B) \in S$ $w_A = w_B$}
begin
compute $|w_A|$ for each nonterminal A;
if there is $(A, B) \in S$ such that $|w_A| \neq |w_B|$ **then return** false;
$(A_1, \ldots, A_n)$:=sort nonterminals in descending order according to $|w_A|$;
rel:=$\bigcup_{(A,B)\in S}(A, B, 0)$;
for i:=1 **to** n **do**
begin rel:=*Split*(A_i,rel); rel:=*Compact*(rel) **end**;
{there are no nonterminals in triples of rel}
if $\exists(a, b, 0) \in$ suffix and $a \neq b$ **then return** false **else return** true
end.

Theorem 12. *The worst-case performance of the algorithm Test is polynomial with respect to the size of input.*

Proof. Let rel_i be the value of the variable *rel* before the i-th iteration of the algorithm. The correctness of the algorithm is a consequence of Lemma 7 and Lemma 10. To prove the polynomial worst-case performance of the algorithm it is enough to prove that the $\#sufsub(rel_i)$ can be polynomially bounded. Initially, $\#subword(rel_0) = 0$ and $\#suffix(rel_0) = |S| \leq n^2$. By Lemma 8 we have $\#suffix(rel_i) \leq (2n+1)n^2$ for $i \geq 1$. Since the operation *Compress* do not touch subword triples we have $\#subword(rel_i) \leq \#subword(rel_{i-1}) + 3\#suffix(rel_{i-1})$ for $i \geq 1$. Solving this recurrence we obtain $\#subword(rel_i) \leq 3i(2n+1)n^2 \leq 3n(2n+1)n^2$ and finally $\#sufsub(rel_i) \leq (3n+1)(2n+1)n^2$.

Theorem 13. *Testing of a set of equivalences between words from a set of words defined by a grammar G can be done in polynomial time.*

As a consequence of the above theorem we have

Theorem 14. *Testing mophism equivalence on a context-free language can be done in polynomial time*

Proof. To compare words $f(w)$, $g(w)$ defined by grammars G_f, G_g consider the grammar $G_f \cup G_g$ and the one element set of pairs of nonterminals $\{(A_f, A_g)\}$ where A_f, A_g are starting symbols of grammars G_f, G_g, respectively.

Because of shortage of space two other applications of algorithm Test are presented in examples.

Example 3. Using algorithm *Test* we can compare first n elements of two sequences of words which are defined by recurrence formulae of form:
$f_{n+s+1} = F(f_n, \ldots, f_{n+s})$ where F is a composition of concatenations and s is a constant. For example to compare first n elements of sequences $f_1 = a$, $f_2 = b$, $f_{k+2} = f_{k+1}f_k$ for $1 \leq k \leq n-2$ and $f'_1 = a$, $f'_2 = b$, $f'_{k+2} = f'_k f'_{k+1}$ for $1 \leq k \leq n-2$ we consider a grammar
$F_1 \to a$, $F_2 \to b$, $F_{k+2} \to F_{k+1}F_k$ for $1 \leq k \leq n-2$
$F'_1 \to a$, $F'_2 \to b$, $F'_{k+2} \to F'_k F'_{k+1}$ for $1 \leq k \leq n-2$
and a set of pairs $\{(F_k, F'_k) : 1 \leq k \leq n\}$.

Example 4. A D0L sequence (w, f, Σ) is a sequence of words w, $f(w)$, $f^2(w)$, ...where w is a starting word, $f : \Sigma^* \to \Sigma^*$ is a morphism and Σ is an n letter alphabet. A well known $2n$ conjecture for D0L sequences states that if two D0L sequences coincide on first $2n$ elements then they are the same. Up to now it is not known whether or not the conjecture holds true. If it is true however then our algorithm can test in polynomial time whether or not two D0L sequences are identical. Consider for example two D0L sequences $(w, f, \{a, b\})$, $(w, g, \{a, b\})$ where $w = aba$, $f(a) = ab$, $f(b) = ba$ and $h(a) = aab$, $h(b) = baa$. In the construction of a grammar to our algorithm we use the identities $f^{k+1}(a) = f^k(a)f^k(b)$, $f^{k+1}(b) = f^k(b)f^k(a)$, $h^{k+1}(a) = h^k(a)h^k(a)h^k(b)$, $h^{n+1} = h^k(b)h^k(a)h^k(a)$. The grammar looks as follows
$F_a^{k+1} \to F_a^k F_b^k$, $F_b^{k+1} \to F_b^k F_a^k$, $F_{aba}^k \to F_a^k F_b^k F_a^k$ for $0 \leq k \leq 2n-1 = 3$,
$H_a^{k+1} \to H_a^k H_a^k H_b^k$, $H_b^{k+1} \to H_b^k H_a^k H_a^k$, $H_{aba}^k \to H_a^k H_b^k H_a^k$ for $0 \leq k \leq 3$,
$F_a^0 \to a$, $F_b^0 \to b$, $H_a^0 \to a$, $H_b^0 \to b$,
and a set of pairs to test is $\{(F_{aba}^k, H_{aba}^k) : 0 \leq k \leq 2n-1 = 3\}$.

Acknowledgements

The author would like to thank Prof. Wojciech Rytter from Instytut Informatyki UW for useful comments to the paper.

References

1. J. Aho, J. Hopcroft, J. Ullman, "The design and analysis of computer algorithms", Addison-Wesley, 1974.
2. J. Albert, K. Culik II, J. Karhumäki, Test sets for context-free languages and algebraic systems of equations, *Inform. Control* **52**(1982), 172–186.
3. J. Jaffar, Minimal and complete word unification, *JACM* **37**(1), 47–85.
4. S. Jarominek, J. Karhumäki, W. Rytter, Efficient construction of test sets for regular and context-free languages, *Theoret. Comp. Science* (to appear).
5. J. Karhumäki, W. Plandowski, W. Rytter, Polynomial size test sets for context-free languages, *JCSS* (to appear).
6. J. Karhumäki, W. Plandowski, W. Rytter, Polynomial size test sets for context-free languages, *in* Proceedings of ICALP'92, Lect. Notes in Comp. Science 623 (1992), 53–64.
7. A. Koscielski, L. Pacholski, Complexity of unification in free groups and free semigroups, *in* Proc. 3st Annual IEEE Symposium on Foundations of Computer Science, Los Alamitos 1990, 824–829.
8. M. Lothaire, "Combinatorics on words", Addison-Wesley Publishing Company, Massachussets, 1983.
9. G.S. Makanin, The problem of solvability of equations in a free semigroup, *Math. USSR Sbornik* **32**, 2(1977), 129–198.
10. J.P. Pecuchet, Equations avec constantes et algorithme de Makanin, These de doctorat, Laboratoire d'informatique, Rouen, 1981.
11. K.U. Schultz, Makanin's algorithm for word equations - two improvements and a generalization, CS Report 91–39, Centrum für Informations und Sprachverarbeitung, University of Munique, 1991.