Algorithms Based on Finite Automata for Testing of Z-codes

Dang Quyet Thang, Nguyen Dinh Han, Phan Trung Huy

Algorithms Based on Finite Automata for Testing of *Z*-codes*

Dang Quyet Thang[1], Nguyen Dinh Han[2], Phan Trung Huy[3]

[1] Nam Dinh University of Technology and Education, Vietnam
thangdgqt@gmail.com
[2] Hung Yen University of Technology and Education, Vietnam
hannguyen@utehy.edu.vn
[3] Hanoi University of Science and Technology, Vietnam
huypt-fami@mail.hut.edu.vn huyfr2002@yahoo.com

Abstract. In this paper, we propose an algorithm to decide whether a regular language recognized by finite automata is a Z-code or not. This algorithm has time complexity $O(n^4)$ for the general case of non-deterministic automata, $O(n^2)$ for the restricted case of deterministic automata, where n is the number of states of finite automata.

Keywords: deterministic automata, bipolar, quadratic algorithm, *Z*-code.

1 Introduction

The *bi-infinite* words play an important part in research infinite behaviors of system, logical models, formal dynamic systems, games and new code construction ... In coding processes, especially in transmission environment such as local networks, internet.. there are various new kinds of encoded messages that we do not know its starting and ending points, therefore its content can not be decoded. Using Z-codes for *bi-infinite* words (message), we can determine the encoding and decoding processes for any bi-infinite message by using some kind of Z-codes... Research *Z*-codes in formal languages, specially in theory of codes has been one of main subjects in many works [2-4,6-8,11,12], etc., which showed the interesting role of Z-codes. A very basic problem is to test wherether or not a language of finite words is a Z-code, specially, when the input is a regular language recognized by finite automata. The techniques to solve this problem provide effective methods to develop research on the related areas of codes, finite graphs and automata. The testing algorithm for *Z*-codes for the case of finite languages is given in [9]. However, for the general case of regular languages, such an algorithm is not known and this is the subject of this paper.

* This work is supported by Vietnamese National Foundation for Science & Technology Development (NAFOSTED)

Here we introduce a new testing algorithm for *Z*-codes with time complexity $O(n^2)$ when the input is deterministic finite automaton, and it is $O(n^4)$ when the input is non-deterministic finite automaton.

In Section 2, we recall some basic notions. In Section 3 we present some algorithms to set up a kind of extended automata. These automata permit us to establish the main result - a new testing algorithm for *Z*-codes in Section 4.

2 Preliminaries

We recall some notions and notations (see [1,2,6,7]). Let Σ be an alphabet. As usual, Σ^* is the free monoid generated by Σ whose elements are called *finite words*. We denote by ε the empty word. We call a nonempty word w *primitive* if it is not a proper power of any word. Set $\Sigma^+ = \Sigma^* - \{\varepsilon\}$. A subset of Σ^* is called a *language.*

In the following, our consideration is mainly based on the notion of infinite words. Let ${}^N\Sigma$, Σ^N, Σ^Z be the sets of left infinite, right infinite and bi-infinite words on Σ respectively. For a language L of Σ^*, we denote ${}^\omega L$, L^ω and ${}^\omega L^\omega$ the left infinite, the right infinite and the bi-infinite product of nonempty words of L respectively.

Factorizations on L of left or right infinite word are understood customarily (see [2,7]), but factorizations of a bi-infinite word need a special treatment as follows. Let $w \in \Sigma^Z$ be in the form: $w = \cdots a_{-2}a_{-1}a_0a_1a_2\cdots$ with $a_i \in \Sigma$. An *L-factorization* of the bi-infinite word w is a strictly increasing function $\mu : Z \to Z$ satisfying $x_i = a_{\mu(i)+1}\cdots a_{\mu(i+1)} \in L$ for all $i \in Z$. Two *L*-factorizations μ and λ are said to be *equal*, denoted by $\mu = \lambda$ if there is $t \in Z$ such that $\lambda(i+t) = \mu(i)$ for all $i \in Z$. Otherwise, μ and λ are *distinct*, denoted by $\mu \neq \lambda$. It is easy to verify that $\mu \neq \lambda$ if and only if $\mu(Z) \neq \lambda(Z)$, or equivalently, there exists a word $u \in \Sigma^+$, two bi-infinite sequences of words of L: $\cdots, x_{-2}, x_{-1}, x_0, x_1, x_2, \cdots$ and $\cdots, y_{-2}, y_{-1}, y_0, y_1, y_2, \cdots$ such that:

$$\cdots x_{-2}x_{-1}u = \cdots y_{-1}y_0,\ |u| \le |x_0|,\ x_0x_1x_2\cdots = uy_1y_2\cdots,\ |u| \le |y_0|, \text{ with } u \neq x_0 \text{ or } u \neq y_0.$$

If every right infinite word of Σ^N has at most one factorization on L then L is said to be an *N-code* (see [7]). Analogously, if every left infinite word of ${}^N\Sigma$ possesses this property, we call L an $\overline{N}$*-code.* Obviously, L is an *N*-code if and only if $\overline{N} = \{\overline{x} : x \in L\}$ is an $\overline{N}$*-code,* where if $x = a_1a_2\cdots a_n$ then $\overline{x} = a_n\cdots a_2a_1$.

Definition 1. A language L of Σ^+ is a *Z-code* if all *L*-factorizations of every bi-infinite word on L are equal.

Example 1. Every singleton $\{u\}$ is always both an *N*-code and an $\overline{N}$-code but it is a Z-code if and only if u is primitive. The language $L=\{ab, ba\}$ is both an *N*-code and

an $\overline{N}$ -code, but it is not a Z-code since the *...ababab...* has two *L*-factorizations ...(*ab*)(*ab*)(*ab*)... and ...(*ba*)(*ba*)(*ba*)..., which are verified directly to be distinct.

A *finite automaton* over Σ is a 5-tuple $\boldsymbol{A}=(Q,\Sigma,E,I,F)$, where Q is a finite set of states, $E\subseteq Q\times\Sigma\times Q$ is a non-empty set of arcs, $I\subseteq Q$ is called *a set of initial states*, $F\subseteq Q$ is called *a set of terminal states*. Each arc $e\in E$ is a tuple $e=(p,a,q)$: *e starts* from *p*, *ends* at *q* and its label is *a*. We also say that *e leaves p for q*. A finite automaton $\boldsymbol{A}$ is called *deterministic* if $\mathrm{Card}(I)\geq 1$ and for any $q\in Q$, $a\in\Sigma$, there is at most one arc that leaves q with the label a. A sequence of arcs $\pi=e_1e_2...e_k\in E^*$, where $e_1=(p_0,a_1,p_1),...,e_k=(p_{k-1},a_k,p_k)$ is called a *path* from p_0 to p_k. The word $w=a_1a_2...a_k$ is the label of π. A path π is called a *successful path* if its start state $p_0\in I$ and its end state $p_k\in F$. In this case, we say that the word as its label is *recognized* by $\boldsymbol{A}$. The set of all words recognized by $\boldsymbol{A}$ is called the language recognized by $\boldsymbol{A}$, denoted by $L(\boldsymbol{A})$.

A *directed graph* is a couple $G=(V,E)$, where V is a set of vertices and E is a set of arcs, each arc is an ordered pair of vertices: $e=(u,v)$, $u,v\in V$. If all V,E are finite we call G finite. A path from a vertex u to v consists of a sequence of vertices $x_0,...,x_n$, where $u=x_0$, $v=x_n$, $(x_i,x_{i+1})\in E$, $i=0,...,n-1$.

3 Extension of Finite Automata

We consider a kind of extended automata to establish a testing algorithm for *Z*-codes.

3.1 Bipolar Automat

A (non-deterministic) finite automaton $\boldsymbol{A}$ is called a *bipolar* automaton if $\boldsymbol{A}$ has only one initial state and one terminal state, and there is no arc starting from the terminal state, there is no arc ending at the initial state.

Given a finite automaton $\boldsymbol{A}=(Q,\Sigma,E,I,F)$ recognizing the language $L\subseteq\Sigma^+$. From $\boldsymbol{A}$, we construct a bipolar automaton $\boldsymbol{A}'=(Q',\Sigma,E',I',F')$ recognizing the same L as follows.

(*i*) Choose two new states $s,f\notin Q$, $s\neq f$, set $Q'=Q\cup\{s,f\}$. set $I'=\{s\}$, $F'=\{f\}$.

(*ii*) Set $E'=E_1\cup\{(s,a,q)|(p,a,q)\in E_1,p\in I\}$, where $E_1=E\cup\{(p,a,f)|(p,a,q)\in E,q\in F\}$.

For simplicity, we denote $\boldsymbol{A}'=(Q',\Sigma,E',s,f)$, and call s the in-polar, f the out-polar. The algorithm to construct $\boldsymbol{A}'$ from $\boldsymbol{A}$ is denoted by $\boldsymbol{A}_D$ has time complexity$O(|Q|+|E|)$.

Given a bipolar automaton $\boldsymbol{A}$ recognizing the language $L\subseteq\Sigma^+$, we can construct an *extended* automaton $\boldsymbol{A}'$ that recognizes L^ω (Büchi' type) by adding an ε-arc which starts at the out-polar, ends at the in-polar of $\boldsymbol{A}$. We also denote $\boldsymbol{A}'$ by $\mathrm{Ex}(\boldsymbol{A}_D)$.

Given an extended automaton $\boldsymbol{A}=(Q,\Sigma,E,s,f)$ recognizing a language L^ω, we construct a *reversed* automaton $\boldsymbol{A}'=(Q',\Sigma,E',s',f')$, with $Q=Q'$,$s'=f$, $f'=s$, E' is the set of

reversed arcs of E. Then $\boldsymbol{A}$' recognizes the language $\overline{L}^{\omega}$. The algorithm to construct $\boldsymbol{A}$' can be expressed by a function Reverse($\boldsymbol{A}$) with time complexity $O(|Q|+|E|)$.

Remark 1. Let $\boldsymbol{A}=(Q,\Sigma,E,I,F)$ be a non-deterministic finite automaton and let $c=|\Sigma|$, $n=|Q|$, $m=|E|$. Then, (*i*) if $\boldsymbol{A}'=\boldsymbol{A}_D$ then $L(\boldsymbol{A})=L(\boldsymbol{A}')$. (*ii*) in special cases, if $\boldsymbol{A}$ is deterministic then for any $q\in Q$, there are at most c arcs leaving q. Hence, $\boldsymbol{A}$ has at most $m=n.c$ arcs. When c can be seen as a constant, the time complexity of setting up $\boldsymbol{A}_D$ is $O(n)$. The number of states of $\boldsymbol{A}_D$, and $\mathrm{Ex}(\boldsymbol{A}_D)$ is at most n+2. The number of arcs of $\boldsymbol{A}_D$ is at most $2m+2c=2nc+2c$ and the number of arcs of $\mathrm{Ex}(\boldsymbol{A}_D)$ is at most $2nc+2c+1$.

3.2 Products of Automata

In this paper we need to construct a new automaton $\boldsymbol{A}=(Q,\Sigma,E,s,f)$ as a product of two automata (see [5]) as follows. Given two extended or bipolar automata $\boldsymbol{A}_1=(Q_1,\Sigma,E_1,s_1,f_1)$ and $\boldsymbol{A}_2=(Q_2,\Sigma,E_2,s_2,f_2)$, where $Q\subseteq Q_1\times Q_2$, E is defined as follows:

(*i*) $\forall(q_1,a,p_1)\in E_1,\ \forall(q_2,a,p_2)\in E_2,\ a\in\Sigma \Rightarrow ((q_1,q_2),a,(p_1,p_2))\in E$

(*ii*) $\forall(q_1,\varepsilon,p_1)\in E_1,\ \forall(q_2,\varepsilon,p_2)\in E_2 \Rightarrow ((q_1,q_2),\varepsilon,(p_1,p_2))\in E$

(*iii*) $\forall(q_1,\varepsilon,p_1)\in E_1,\ \forall(q_2,a,p_2)\in E_2,\ a\in\Sigma \Rightarrow ((q_1,q_2),\varepsilon,(p_1,q_2))\in E$

(*iv*) $\forall(q_1,a,p_1)\in E_1,\ \forall(q_2,\varepsilon,p_2)\in E_2,\ a\in\Sigma \Rightarrow ((q_1,q_2),\varepsilon,(q_1,p_2))\in E$

(*v*) E has only arcs defined by above four cases.

Set $s=(s_1,s_2)$, $f=(f_1,f_2)$. The algorithm is presented by a function named Prod($\boldsymbol{A}_1$,$\boldsymbol{A}_2$).

Remark 2. (*i*) Similar to Mohri's analysis in [5], we see that the algorithm has time complexity $O((|Q_1|+|E_1|)(|Q_2|+|E_2|))$. By Remark 1, if $\boldsymbol{A}_1$, $\boldsymbol{A}_2$ are deterministic then the algorithm has time complexity $O(|Q_1||Q_2|)$. (*ii*) Let $\boldsymbol{A}_1=(Q_1,\Sigma,E_1,s_1,f_1)$ be an extended automaton that recognizes L^{ω}. In the automaton Prod($\boldsymbol{A}_1$,$\boldsymbol{A}_1$), the label of a path connecting two consecutive states (f_1,q_i) and (f_1,q_j) (or (p_i,f_1) and (p_j,f_1), or (s_1,q_i) and (f_1,q_j), or (p_i,s_1) and (p_j,f_1)) is the word of L.

3.3 Union-product of Automata

Given two extended automata $\boldsymbol{A}_1=(Q_1,\Sigma,E_1,s_1,f_1)$ and $\boldsymbol{A}_2=(Q_2,\Sigma,E_2,s_2,f_2)$, then the *union-products* of $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ is defined as ProdUni($\boldsymbol{A}_1$,$\boldsymbol{A}_2$)= $(Q,\Sigma,E,I,\{(f_1,f_2)\})$, where $I=\{(f_1,q)\,|\,\forall q\in Q_2, q\neq f_2\wedge q\neq s_2\}$, $Q\subseteq Q_1\times Q_2$, and E is defined according to the rules from (*i*) to (*v*) in section 3.2. We add the initial state (s,s) into Q, and then add the arcs $\{((s,s),\varepsilon,(f_1,q))\,|\,\forall q\in Q_2, q\neq f_2\wedge q\neq s_2\}$ into E. Then we have the union-product automaton with one initial state ProdUni($\boldsymbol{A}_1$,$\boldsymbol{A}_2$)=$(Q,\Sigma,E,(s,s),(f_1,f_2))$.

Remark 3. (*i*) Similar to the algorithm designing the product of automata, building ProdUni($\boldsymbol{A}_1$,$\boldsymbol{A}_2$) has time complexity $O((|Q_1|+|E_1|)(|Q_2|+|E_2|))$. By Remark 1, if $\boldsymbol{A}_1$, $\boldsymbol{A}_2$ are deterministic then this algorithm has a time complexity $O(|Q_1||Q_2|)$.

(*ii*) Let $\boldsymbol{A}_1=(Q_1,\Sigma,E_1,s_1,f_1)$ be an extended automaton that recognizes L^{ω}. In the automaton ProdUni($\boldsymbol{A}_1$,$\boldsymbol{A}_1$), the label of the path connecting two consecutive states (f_1,q_i) and (f_1,q_j) (or (p_i,f_1) and (p_j,f_1)) is a word of L.

4 Algorithms for Testing of *Z*-codes

At first we need to solve an extra problem on graph which is interesting by itseft.

4.1 A problem on Finite Graphs

Given a directed finite graph $G=(V,E)$ and two vertices: s is the *initial vertex* and f is the *terminal vertex* ($s\neq f$). Let $U\subseteq V$ be a set of vertices which are called *up-keys*, $D\subseteq V$ be a set of vertices which are called *down-keys* such that $U\cap D=\emptyset$ (the vertices s and f are not in $U\cup D$). The rest vertices are called *non-key* vertices.

Given a finite path $\pi=v_1,v_2,...,v_k$, $v_i\in V$, $1\leq i\leq k$ ($s=v_1$) in G. Then, (*i*) if there exist i,j, $1<i<j<k$ such that $v_i\in U$, $v_j\in D$ and $v_k=v_i$ then π is called *a path of type* 1.

(*ii*) if there exists i, $1<i<k$ such that $v_i\in U$ and $v_k=f$ then π is called *a path of type* 2.

Problem 1. Let $G=(V,E)$ be a graph defined as above. Set up an algorithm to verify if there exists any path of type 1 in G and if there exists any path of type 2 in G.

To solve this problem, we need to construct a graph $G'=(V',E')$ from G by using a "graph-copy" technique as follows:

(*i*) for each $v\in V$, create three vertices $(v,1),(v,2),(v,3)$ as copies of v and put to V'.

(*ii*) for each arc $(u,v)\in E$, create three arcs $((u,1),(v,1)),((u,2),(v,2))$ and $((u,3),(v,3))$ as copies of (u,v) and put to E'. Moreover, if $u\in U$, create a new arc $((u,1),(v,2))$ and update to E'. If $u\in D$, create a new arc $((u,2),(v,3))$ and update to E'.

The algorithm to construct G' from G can be expressed by a function XCopy(G) with time complexity $O(|V|+|E|)$.

Remark 4. (*i*) By G' constructed as above, we have: $|V'|=3n$, $|E'|\leq 5m$ with $|V|=n$, $|E|=m$. (*ii*) The set of vertices of the type (v,k) in G' generates the subgraph G_k for each $k=1,2,3$. Each subgraph G_k is isomorphic to G. Hence G' is a version extended from the union of G_1,G_2,G_3. There are some arcs from G_1 to G_2, but not any in the reversed direction. It is similar from G_2 to G_3.

The following theorem describes the meaning of G'

Theorem 1. *Let $G=(V,E)$ be a finite graph defined as above and $G'=XCopy(G)$.*

(*i*) *There exists a path of type* 1 *in G if and only if there exists a path π in G' that starts at the vertex* $(s,1)$ *and ends at* $(v,3)$*, where $v\in U$ and* $(v,1)\in\pi$.

(*ii*) *There exists a path of type* 2 *in G if and only if there exists a path π in G' that starts at the vertex* $(s,1)$ *and ends at* $(f,2)$.

Proof. (*i*) ($\Rightarrow$) There exists a path of type 1 in G: $u_1,...,u_i,...,u_j,...,u_k$ ($s=u_1$). We have $1<i<j<k$ such that $u_i\in U$, $u_j\in D$ and $u_i=u_k$. According to the function constructing G', there exists a path π, $(u_1,1),...,(u_i,1),(u_{i+1},2),...,(u_j,2),(u_{j+1},3),...,(u_k,3)$ where $(s,1)=(u_1,1),(u_k,3)=(v,3)$. It is obviously that $v\in U$ and $(v,1)\in\pi$.

($\Leftarrow$) There exists a path π in G' from $(s,1)$ to $(v,3)$, where $v\in U$ and $(v,1)\in\pi$. According to the function constructing G', π can be written as:

$(u_1,1),...,(u_i,1),(u_{i+1},2),...,(u_j,2),(u_{j+1},3),...,(u_k,3)$, where $(s,1)=(u_1,1),(u_k,3)=(v,3)$, $1<i<j<k$, $u_i \in U$, $u_j \in D$ and $u_k=u_i$. Then, in G, there is a path of type 1: $u_1,...,u_i,u_{i+1},...,u_j,u_{j+1},...,u_k$ where $s=u_1,u_k=v$, $1<i<j<k$, $u_i \in U$, $u_j \in D$ and $u_k=u_i$.

(*ii*) It can be easily implied from the function constructing G'. □

To test whether there exists any path of type 1 in G, we use an array *mark* as follows: a vertex (u,i) is colored WHITE, (or *mark*$[(u,i)]$=WHITE) to show that (u,i) has not been visited. For a vertex is considered, if its type is $(u,1)$ with $u \in U$, then it is colored BLUE, otherwise it is colored GREY. Whenever the considered vertex $(u,1)$ is colored BLUE, a corresponding vertex $(u,3)$ will be colored BLUE. This guarantees the fact that if we have a path in G' starting from $(s,1)$ and ending at $(u,3)$ which is colored BLUE then we also have a path of type 1 in G. For testing a path of type 2, we need only a path from $(s,1)$ to $(f,2)$ without using this coloring technique. A vertex is colored BLACK if it is not included in any further visiting process.

In general, the algorithm can be described as follows: (*i*) Initially, all vertices of G' are colored WHITE. (*ii*) We call a recursive function *visit* to visit all vertices of G' by using the coloring technique mentioned above. This function is modified from the DFS (Depth First Search) algorithm in [10] as follows.

```
Function int Visit(graph G', vertex (u,i),int x)
//x=1 function Visit detects path of type 1, type 2.
//x=0 function Visit only detects path of type1.
1.  if i=1 and upkey[u]=1 then
      mark[(u,1)]=mark[(u,3)]=BLUE else mark[(u,i)]=GREY
2.  for each arc ((u,i),(v,j))∈E' do
      if mark[(v,j)]=BLUE and j=3 then return 1;
      if (v,j)=(f,2) and x=1 then return 2;
      if mark[(v,j)]=WHITE then
        if Visit(G',(v,j),x)!=0 then
                  return Visit(G',(v,j),x);
3.  mark[(u,i)]=BLACK
    if i=1 and mark[(u,3)]=BLUE then mark[(u,3)]=WHITE
4.  return 0.

Function int ContainsCycle(graph G', vertex (u,i), int x)
1.  for each vertex (u,1)∈V' do
      mark[(u,1)]=mark[(u,2)]=mark[(u,3)]=WHITE
2.  return Visit(G',(u,i),x);
```

Remark 5. The algorithm ContainsCycle detecting any paths of type 1, 2 has its time complexity $O(|V|+|E|)$.

4.2 A New Algorithm for Testing of Z-codes

In this part, we present the main results of this paper. Let $L \subseteq \Sigma^*$ be a regular language recognized by a given finite automaton $\boldsymbol{A}=(Q,\Sigma,E,I,F)$ which is generally non-deterministic. We consider step by step the following cases.

(*i*) if $\varepsilon \in L$ then L is not a Z-code. To check whether $\varepsilon \in L$ reduces to determine whether $I \cap F$ is emptyset. We present I, F by two arrays as follows: $InI(q)=1 \Leftrightarrow q \in I$ and $InF(q)=1 \Leftrightarrow q \in F$. Therefore, testing whether $I \cap F$ is emptyset can be done by an algorithm Epsilon($\boldsymbol{A}$), with time complexity $O(n)$, $n=|Q|$.

(*ii*) if $\varepsilon \notin L$ then $L \subseteq \Sigma^+$. Suppose that L is not a Z-code. Then there exists a word $w \in \Sigma^z$ which has two different L-factorizations. It can be one of four types as shown in Figure 1. Hence, verifying whether L is a Z-code reduces to testing whether there exists such a word w. We construct $\boldsymbol{A}_1=\mathrm{Ex}(\boldsymbol{A}_D)=(Q_1,\Sigma,E_1,s_1,f_1)$, $\boldsymbol{A}_2=\mathrm{Reverse}(\boldsymbol{A}_1)=(Q_2,\Sigma,E_2,s_2,f_2)$ and from these automata we have:

Prod($\boldsymbol{A}_1,\boldsymbol{A}_1$) induces a directed graph G_1 with the initial vertex (s_1,s_1), the terminal vertex (f_1,f_1), the set of up-keys $U_1=\{(f_1,q)\in Q_1\times Q_1 |\ q\neq f_1 \wedge q\neq s_1\}$, the set of down-keys $D_1=\{(p,f_1)\in Q_1\times Q_1 |\ p\neq f_1 \wedge p\neq s_1\}$, and each arc of Prod($\boldsymbol{A}_1,\boldsymbol{A}_1$) is an arc of G_1.

Prod($\boldsymbol{A}_2,\boldsymbol{A}_2$) induces a directed graph G_2 with the initial vertex (s_2,s_2), the terminal vertex (f_2,f_2), the set of up-keys $U_2=\{(f_2,q)\in Q_2\times Q_2 |\ q\neq f_2 \wedge q\neq s_2\}$, the set of down-keys $D_2=\{(p,f_2)\in Q_2\times Q_2 |\ p\neq f_2 \wedge p\neq s_2\}$ and each arc of Prod($\boldsymbol{A}_2,\boldsymbol{A}_2$) is an arc of G_2.

ProdUni($\boldsymbol{A}_1,\boldsymbol{A}_1$) induces a directed graph G_3 with the initial vertex (s,s), the terminal vertex (f_1,f_1), the set of up-keys $U_3=\{(f_1,q)\in Q_1\times Q_1 | q\neq f_1 \wedge q\neq s_1\}$, the set of down-keys $D_3=\{(p,f_1)\in Q_1\times Q_1 | p\neq f_1 \wedge p\neq s_1\}$ and each arc of ProdUni($\boldsymbol{A}_1,\boldsymbol{A}_1$) is an arc of G_3.

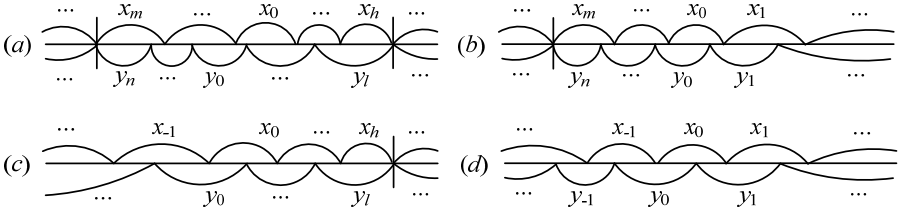

Fig. 1. Four types of two different L-factorizations of w

For the case $\varepsilon \notin L$, we establish the following result

Theorem 2. *Let $L \subseteq \Sigma^+$ be a language recognized by a finite automaton $\boldsymbol{A}$, let G_1, G_2, G_3 be defined as above. L is a Z-code if and only if four following conditions hold*

(*i*) *there is no any paths of type* 2 *in the graph* G_1;
(*ii*) *there is no any paths of type* 1 *in the graph* G_1;
(*iii*) *there is no any paths of type* 1 *in the graph* G_2;
(*iv*) *there is no any paths of type* 1 *in the graph* G_3.

Proof. ($\Rightarrow$) We assume by a contradiction that *L is a Z-code* and one of the conditions (*i*), (*ii*), (*iii*) or (*iv*) does not hold.

(*i*) Suppose that there exists a path of type 2 in G_1, we consider the label of this path: $(p_1,q_1)\xrightarrow{x}(p_i,q_i)\xrightarrow{y}(p_k,q_k)$, where $(s_1,s_1)=(p_1,q_1),(p_i,q_i)=(f_1,q_i)\in U_1$, $1<i<k$ and $(p_k,q_k)=(f_1,f_1)$. Hence we have a path π: $(p_1,q_1)\xrightarrow{x}(f_1,q_i)\xrightarrow{y}(f_1,f_1)$. By Remark 2, we have $v=xy\in L^+$. With any $u\in{}^{\omega}L$, $z\in L^{\omega}$, then the word $w=uvz\in{}^{\omega}L^{\omega}$ admits two different *L*-factorizations. Therefore, *L* is not a *Z*-code. This contradicts the assumption.

(*ii*) Suppose that there exists a path of type 1 in G_1, we consider the label of this path: $(p_1,q_1)\xrightarrow{x}(p_i,q_i)\xrightarrow{y}(p_j,q_j)\xrightarrow{z}(p_k,q_k)$, where $(s_1,s_1)=(p_1,q_1)$, $(p_i,q_i)=(f_1,q_i)\in U_1$, $(p_j,q_j)=(p_j,f_1)\in D_1$, $1<i<j<k$ and $(p_k,q_k)=(p_i,q_i)$. Hence, we have a right infinite path π:

$$(p_1,q_1)\xrightarrow{x}(f_1,q_i)\xrightarrow{y}(p_j,f_1)\xrightarrow{z}(f_1,q_i)\xrightarrow{y}(p_j,f_1)\xrightarrow{z}(f_1,q_i)\xrightarrow{y}\cdots$$

By Remark 2, we have $v=xyzyzyz\cdots\in L^{\omega}$. With any $u\in{}^{\omega}L$, then the word $w=uv\in{}^{\omega}L^{\omega}$ admits two different *L*-factorizations. Therefore *L* is not a *Z*-code. This contradicts the assumption.

(*iii*) Suppose that there exists a path of type 1 in G_2, we consider the label of this path: $(p_1,q_1)\xrightarrow{\bar{x}}(p_i,q_i)\xrightarrow{\bar{y}}(p_j,q_j)\xrightarrow{\bar{z}}(p_k,q_k)$, where $(s_2,s_2)=(p_1,q_1)$, $(p_i,q_i)=(f_2,q_i)\in U_2$, $(p_j,q_j)=(p_j,f_2)\in D_2$, $1<i<j<k$ and $(p_k,q_k)=(p_i,q_i)$. Hence we have a right infinite path π:

$$(p_1,q_1)\xrightarrow{\bar{x}}(f_2,q_i)\xrightarrow{\bar{y}}(p_j,f_2)\xrightarrow{\bar{z}}(f_2,q_i)\xrightarrow{\bar{y}}(p_j,f_2)\xrightarrow{\bar{z}}(f_2,q_i)\xrightarrow{\bar{y}}\cdots$$

By Remark 2, we have $t=\bar{x}\ \bar{y}\ \bar{z}\ \bar{y}\ \bar{z}...\in\overline{L}^{\omega}$. Then, label *t* of π admits two different factorizations on $\overline{L}$. Correspondingly, we have *v*=... *zyzyx* $\in {}^{\omega}L$ with two differrent factorizations on *L*. With any $u\in L^{\omega}$, then the word $w=vu\in{}^{\omega}L^{\omega}$ admits two different *L*-factorizations. Therefore, *L* is not a *Z*-code. This contradicts the assumption.

(*iv*) Suppose that there exists a path of type 1 in G_3, we consider the label of this path: $(p_1,q_1)\xrightarrow{x}(p_i,q_i)\xrightarrow{y}(p_j,q_j)\xrightarrow{z}(p_k,q_k)$, where $(s,s)=(p_1,q_1)$, $(p_i,q_i)=(f_1,q_i)\in U_3$, $(p_j,q_j)=(p_j,f_1)\in D_3$, $1<i<j<k$ and $(p_k,q_k)=(p_i,q_i)$. Hence, we have a bi-infinite path π:

$$\cdots(f_1,q_i)\xrightarrow{y}(p_j,f_1)\xrightarrow{z}(f_1,q_i)\xrightarrow{y}(p_j,f_1)\xrightarrow{z}(f_1,q_i)\xrightarrow{y}\cdots$$

By Remark 3, we have $w=\cdots yzyzyz\cdots\in{}^{\omega}L^{\omega}$. It implies that *w* admits two different *L*-factorizations. Therefore *L* is not a *Z*-code. This contradicts the assumption.

($\Leftarrow$) We assume by a contradiction that the four conditions (*i*)-(*iv*) hold but *L* is not a *Z*-code. Then there exists a word $u\in\Sigma^+$, two bi-infinite sequences of words of *L*: $\cdots,x_{-2},x_{-1},x_0,x_1,x_2,\cdots$ and $\cdots,y_{-2},y_{-1},y_0,y_1,y_2,\cdots$ such that:

$$\cdots x_{-2}x_{-1}u=\cdots y_{-1}y_0,\ |u|\le|x_0|,\ x_0x_1x_2\cdots=uy_1y_2\cdots,\ |u|\le|y_0|,\text{ with } u\ne x_0 \text{ or } u\ne y_0.$$

We consider four cases (see Figure 1) may be happened as follows:

Case 1: There exist $m \le 0 \le h$, $n \le 0 \le l$ with $m \ne 0$ or $l \ne 0$ (Figure 1.a) such that $v = x_m x_{m+1} \cdots x_h = y_n y_{n+1} \cdots y_l$. Since $v = x_m x_{m+1} \cdots x_h$, $\boldsymbol{A}_1$ has a path π labeled v: $s_1 \xrightarrow{x_m} f_1 \xrightarrow{\varepsilon} s_1 \xrightarrow{x_{m+1}} f_1 \xrightarrow{\varepsilon} s_1 \cdots s_1 \xrightarrow{x_h} f_1$. Similarly, $v = y_n y_{n+1} \cdots y_l$, $\boldsymbol{A}_1$ has a path θ labeled v: $s_1 \xrightarrow{y_n} f_1 \xrightarrow{\varepsilon} s_1 \xrightarrow{y_{n+1}} f_1 \xrightarrow{\varepsilon} s_1 \cdots s_1 \xrightarrow{y_l} f_1$.

Hence, in the graph G_1, there is a path ρ defined by π and θ as follows: $(p_1,q_1),..,(f_1,q_i),..,(p_k,q_k)$ where $(s_1,s_1) = (p_1,q_1)$, $(f_1,q_i) \in U_1$, $(f_1,f_1) = (p_k,q_k)$, $1 < i < k$, or equivalently, G_1 has a path of type 2. This contradicts the condition (*i*).

Case 2: There exist $n,m \le 0$ such that $v = x_m x_{m+1} \cdots = y_n y_{n+1} \cdots$ and there are no $h > m, l > n$ such that $x_h x_{h+1} \cdots = y_l y_{l+1} \cdots$ (Figure 1.b). Since $v = x_m x_{m+1} \cdots$, $\boldsymbol{A}_1$ has a right infinite path π labeled v: $s_1 \xrightarrow{x_m} f_1 \xrightarrow{\varepsilon} s_1 \xrightarrow{x_{m+1}} f_1 \xrightarrow{\varepsilon} s_1 \cdots$ Similarly, $v = y_n y_{n+1} \cdots$, $\boldsymbol{A}_1$ has a right infinite path θ labeled v: $s_1 \xrightarrow{y_n} f_1 \xrightarrow{\varepsilon} s_1 \xrightarrow{y_{n+1}} f_1 \cdots$

Hence, in the graph G_1, there is a right infinite path ρ defined by π and θ as follows: $(p_1,q_1),...,(f_1,q_i),...,(p_j,f_1),...$ where $(s_1,s_1) = (p_1,q_1)$. Since ρ is a right infinite path in the finite graph G_1, there must exist vertices $(p_k,q_k) \in Q_1 \times Q_1$, $(f_1,q_i) \in U_1$, $(p_j,f_1) \in D_1$ such that $(f_1,q_i) = (p_k,q_k)$ with $1 < i < j < k$, or equivalently, G_1 has a path of type 1. This contradicts the condition (*ii*).

Case 3: There exist $h,l \ge 0$ such that $v = \cdots x_{h-1} x_h = \cdots y_{l-1} y_l$ and there are no $m < h, n < l$ such that $\cdots x_{m-1} x_m = \cdots y_{n-1} y_n$ (Figure 1.c). Then, we have $u = \overline{x}_h \overline{x}_{h-1} \cdots = \overline{y}_l \overline{y}_{l-1} \cdots$ with $\overline{x}_h, \overline{x}_{h-1}, \cdots, \overline{y}_l, \overline{y}_{l-1}, \cdots \in \overline{L}$ and there are no $m < h, n < l$ such that $\overline{x}_m \overline{x}_{m-1} \cdots = \overline{y}_n \overline{y}_{n-1} \cdots$. Since $u = \overline{x}_h \overline{x}_{h-1} \cdots$, $\boldsymbol{A}_2$ has a right infinite path π labeled u: $s_2 \xrightarrow{\overline{x}_h} f_2 \xrightarrow{\varepsilon} s_2 \xrightarrow{\overline{x}_{h-1}} f_2 \xrightarrow{\varepsilon} s_2 \cdots$ Similarly, $u = \overline{y}_l \overline{y}_{l-1} \cdots$, $\boldsymbol{A}_2$ has a right infinite path θ labeled u: $s_2 \xrightarrow{\overline{y}_l} f_2 \xrightarrow{\varepsilon} s_2 \xrightarrow{\overline{y}_{l-1}} f_2 \xrightarrow{\varepsilon} s_2 \cdots$

Hence, in the graph G_2, there is a right infinite path ρ defined by π and θ as follows: $(p_1,q_1),...,(f_2,q_i),...,(p_j,f_2),...$ where $(s_2,s_2) = (p_1,q_1)$. Since ρ is a right infinite path in the finite graph G_2, there must exist vertices $(p_k,q_k) \in Q_2 \times Q_2$, $(f_2,q_i) \in U_2$, $(p_j,f_2) \in D_2$ such that $(f_2,q_i) = (p_k,q_k)$ with $1 < i < j < k$, or equivalently, G_2 has a path of type 1. This contradicts the condition (*iii*).

Case 4: There exist no h,l such that $x_h x_{h+1} \cdots = y_l y_{l+1} \cdots$ or $\cdots x_{h-1} x_h = \cdots y_{l-1} y_l$ (Figure 1.d). We have $v = x_m x_{m+1} \cdots = y y_n y_{n+1} \cdots$ with $x_m, x_{m+1}, ..., y_n, y_{n+1}, ... \in L$, $y \in \Sigma^+$ and $y \notin L$. Since $v = x_m x_{m+1} \cdots$, $\boldsymbol{A}_1$ has a right infinite path π labeled v: $s_1 \xrightarrow{x_m} f_1 \xrightarrow{\varepsilon} s_1 \xrightarrow{x_{m+1}} f_1 \xrightarrow{\varepsilon} s_1 \cdots$ Similarly, $v = y y_n y_{n+1} \cdots$, $\boldsymbol{A}_1$ has a right

infinite path θ labeled v: $q_i \xrightarrow{y} f_1 \xrightarrow{\varepsilon} s_1 \xrightarrow{y_n} f_1 \xrightarrow{\varepsilon} s_1 \xrightarrow{y_{n+1}} f_1 \xrightarrow{\varepsilon} s_1 \cdots$ where $q_i \neq s_1, f_1$. Hence, in the graph G_3, there is a right infinite path ρ defined by π and θ as follows: $(p_1,q_1),...,(f_1,q_i),...,(p_j,f_1),...$ where $(s,s)=(p_1,q_1)$. Since ρ is a right infinite path in the finite graph G_3, there must exist vertices $(p_k,q_k) \in Q_1 \times Q_1$, $(f_1,q_i) \in U_3$, $(p_j,f_1) \in D_3$ such that $(f_1,q_i)=(p_k,q_k)$ with $1<i<j<k$, or equivalently, G_3 has a path of type 1. This contradicts the condition (*iv*). These complete the proof of theorem . □

Now we formulate an effective algorithm for testing of *Z*-codes.

Function ZCode(*A*)

Input: $A=(Q,\Sigma,E,I,F)$, $n=|Q|$, $m=|E|$ and $L=L(A)\subseteq\Sigma^*$.
Output: TRUE if L is a Z-code, FALSE otherwise.

```
1.  if Epsilon(A) then return FALSE;
2.  A1=Ex(AD);
3.  G1=Prod(A1,A1);
4.  G=XCopy(G1);
5.  if ContainsCycle(G,((s1,s1),1),1)!=0 then return FALSE
6.  A2=Reverse(A1);
7.  G2=Prod(A2,A2);
8.  G=XCopy(G2);
9.  if ContainsCycle(G,((s2,s2),1),0)==1 then return FALSE
10. G3=ProdUni(A1,A1);
11. G=XCopy(G3);
12. if ContainsCycle(G,((s,s),1),0)==1 then return FALSE
13. return TRUE
```

Time complexity of the algorithm The time complexity for Step 1 is $O(n)$, Steps 2, 6 is $O(n+m)$, Steps 3, 7, 10 is $O((n+m)^2)$, Steps 4, 5, 8, 9, 11, 12 is $O(n^2+m^2)$. Therefore, the time complexity of the whole algorithm is $O(n^4)$ in the case $\boldsymbol{A}$ is non-deterministic, and $O(n^2)$ in the case $\boldsymbol{A}$ is deterministic.

5 Conclusion

Studying on advanced automata models are paid much attention in both theoretic and application aspects. In this paper, we propose an algorithm to decide whether a regular language recognized by a finite automaton is a Z-code or not with time complexity $O(n^4)$ - for the general case of non-deterministic automata, and $O(n^2)$ for the restricted case of deterministic automata. This is a significant problem in terms of theory and practice.

References

1. Berstel, J., Perrin, D.: Theory of Codes. Academic Press Inc, New York (1985)
2. Devolder, J., Latteux, M., Litovsky, I., Staiger, L.: Codes and infinite words. Acta Cybernetica 11(4), 241-256 (1994)
3. Staiger, L.: On infinitary finite length codes. Informatique Théorique et Applications 20(4), 483-494 (1986)
4. Van, D.L.: Contribution to Combinatorics on Words, PhD thesis, Humboldt University, Berlin (1985)
5. Mohri, M., Pereira, F., Riley, M.: Speech recognition with weighted finite-state transducers. Springer, Heidelberg (2007)
6. Devolder, J., Timmerman, E.: Finitary codes for biinfinite words. Informatique Théorique et Applications 26(4), 363-386 (1992)
7. Van, D.L., Lam, N.H., Huy, P.T.: On codes concerning bi-infinite words. Acta Cybernetica 11(1-2), 97-109 (1993)
8. Lam, N.H., Van, D.L.: On a class of infinitary codes. Theoretical Infomatics and Applications 24 , 441-458 (1990)
9. Madonia, M., Salemi, S., Sportelli, T.: A generalization of Sardinas-Patterson algorithm to Z-codes. Theoretical Computer Science 108, 251-270 (1993)
10. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms (3rd ed.). MIT Press and McGraw-Hill (2009)
11. Van, D.L., Thomas, D.G., Subramanian, K.G., Siromoney, R.: Bi-Infinitary Codes. RAIRO Inform. Theor. Appl. 24(1), 67-87 (1990)
12. Restivo, A.: Codes and automata, Lecture Notes in Comput. Sci. 386, 186-198 (1989)