Acta Cybernetica **13** (1997) 23-39.

Server Problems and Regular Languages

B. Csaba [*] G. Dányi [†]

Abstract

The sequences of requests are considered as words over the alphabet of vertices. We assume that the server problem is restricted, meaning that the request words are chosen from a subset of all possible words, i.e. from a language. We define the class *ONLINE* consisting of the languages, for which there exists a 1-competitive satisfying on-line algorithm. Our main result is a sufficient condition for languages to be in *ONLINE* and a construction method of 1-competitive on-line algorithm for the ones, which satisfy that condition. We perform this by characterizing a subclass $ONREG_0$ of the class $ONLINE \cap REG$, where *REG* is the class of regular languages. Moreover, we prove some results, which help to show the on-lineness of certain other (even nonregular) languages and we give sufficient conditions to prove that a language is not on-line.

1 Introduction

The k-server problem is a generalized model of certain scheduling problems as, for instance, multi-level memory paging, disk caching and head motion planning of multi-headed disks (see [MMS]). The paging and caching problems have been studied for a long time. However, server problems are introduced in the 80's (see [ST] and [MMS]).

The k-server problem can be stated as follows. Let $M = (V, \delta)$ be a finite metric space, where $V = \{v_1, \ldots, v_n\}$ are the vertices and δ is the distance function. There are k mobile servers occupy exactly k vertices of M. Repeatedly a request, $v_i \in V$ appears. The request should be satisfied by moving some servers resulting that a server appears on the point v_i. The cost of moving one server from v_i to v_j is $\delta(v_i, v_j)$ and the cost of a satisfaction is the sum of the costs of the taken movements.

[*]Research of this author was supported by the Research Foundation of Hungary under Grant F4204.
Dept. of Computer Science, József Attila University, H-6701 Szeged, P.O.Box 652, Hungary, email: csaba@inf.u-szeged.hu

[†]Research of the author was supported by the Research Foundation of Hungary under Grant F012852 and by the Hungarian Cultural and Educational Ministry under Grant 434/94.
Dept. of Foundations of Computer Science, József Attila University, H-6701 Szeged, P.O.Box 652, Hungary, email: danyi@inf.u-szeged.hu

We assume that the request sequences are finite. The goal is to find an algorithm for M, which can satisfy request sequences with as little'cost as possible.

If an algorithm serves requests immediately without knowing what the future requests will be, then we say that it is on-line. A widely used measure for the performance of an on-line algorithm is the competitive ratio, introduced by [ST]. Denote the optimal cost of the satisfaction of a request sequence x by $\mathrm{opt}(x)$. An on-line algorithm $\mathcal{A}$ is called c-competitive, if there exists a number K such that, for all allowed request sequences x, the total cost $\mathcal{A}(x)$, incurred by $\mathcal{A}$ on x, is at most $c\,\mathrm{opt}(x) + K$.

Obviously, a finite request sequence can be considered as a word over the alphabet V. It has been proved that, if the request sequences can be arbitrarily chosen, that is, any word in V^*, then the competitive ratio of any on-line algorithm is at least k (see [MMS]). However, in practice the request sequences are generated by programs, hence these sequences usually cannot be arbitrary, i.e. they are chosen from a language $L \subseteq V^*$. Then the server problem is said to be restricted. Knowing that language, we may expect to find on-line algorithms with better performance. We can assign competitive ratio to the languages, too. Actually, a language is called c-competitive, if, for any distance function δ, there exists c-competitive on-line algorithm satisfying its request sequences. Observe that the competitive ratio defines a hierarchy of language classes.

In this paper we consider the class $ONLINE$ consisting of 1-competitive languages, called on-line languages. Note that this class is the bottom element of the hierarchy mentioned above. However, $ONLINE$ seems to be very hard to characterize. For instance, it contains languages, which are event not recursively enumerable. For that very reason, we looked for necessary and sufficient conditions a language being in $ONLINE$.

Sufficient conditions can be found, if we consider appropriate subclasses of $ONLINE$. Such subclass can be defined, for example, by intersecting $ONLINE$ with a well known language class. In this paper we choose the class REG of regular languages for this purpose. This class is easy to handle, since the regular languages are recognized by deterministic finite automata. On the other hand, REG is closed for the operations concatenation, union and closure, which corresponds naturally to the programming structures, namely the sequencing, the selection, and the iteration, respectively. Observe that, for any alphabet V, $V^* \in REG$ holds, hence $REG \not\subseteq ONLINE$ follows. We define the subclass $ONREG_0 \subseteq REG$. The class $ONREG_0$ is closed for concatenation, union and closure of singleton languages. Roughly speaking, if we consider programming structures, there cannot be a selection inside an iteration. Our main result is that $ONREG_0 \subseteq ONLINE$. Moreover, we show that how the operations sublanguage construction and letter reduction can help to prove the on-lineness of certain other, even nonregular languages, and we give sufficient conditions to prove that a language is not on-line.

The outline of our paper is as follows. In the second section we introduce the definitions and notations, which are necessary to understand the paper. Moreover, we give some basic results. Our main result can be found in the third section, in which we show that the language class $ONREG_0$ contains on-line languages. In the fourth section we give some sufficient conditions for regular languages not being on-line, and we discuss some other properties of the class *ONLINE*.

We note that there is an other way of studying restricted k-server problems, where the movements of the servers are restricted (see [FK], [BIRS]). This leads to the use of an access graph. A server can be moved to a vertex from an other one immediately, if they are adjacent vertices of the access graph. However, it can be seen that an access graph also defines a language over V, namely the set of satisfiable request sequences.

Acknowledgement. The authors are grateful to P. Hajnal (Dept. of Mathematics, University of Szeged, Hungary) and László Bernátsky (Dept. of Computer Science, University of Szeged, Hungary) for their valuable comments and suggestions.

2 Preliminaries

In this section we introduce the notions and notations which are necessary to understand the paper. Moreover, we recall the preliminaries referred in our proofs from other papers, and give some basic results.

We denote the set of real numbers by $\mathbf{R}$, the set of nonnegative real numbers by $\mathbf{R}^+$ and the set of natural numbers by ω. If H is a set, then $|H|$ denotes its cardinality.

We frequently use the principline of structural induction in our proofs. For more information about inductions see, for example, [W].

2.1 Languages and automata

Words and languages. An *alphabet* V is a finite nonempty set of symbols. The elements of an alphabet are called *letters* and denoted by u and v in this paper.

A *word* w over an alphabet V is a finite sequence $v_1 \dots v_l$ of some letters in V. The *length* of a word w, denoted by $|w|$, is the number of the letters composing w. The *empty string* is denoted by e, thus $|e| = 0$. For $w \in V^* - \{e\}$, we define $\text{first}(w) \in V$ and $\text{last}(w) \in V$ as the first and the last letter of w, respectively.

The *concatenation* $w_1 w_2$ of two words $w_1 = v_1 \dots v_l$ and $w_2 = u_1 \dots u_k$ is the sequence $v_1 \dots v_l u_1 \dots u_k$. We define the powers of a word w as $w^0 = e$ and $w^n = w^{n-1}w$, for any integer $n \geq 1$. We say that a word w_1 is a *prefix* of a word w, and denote this fact by $w_1 \sqsubseteq w$, if there exists a word w_2, called a *suffix* of w, such that $w = w_1 w_2$. We use the symbols w, x and y to denote words in this paper.

The set of all finite words over an alphabet V is denoted by V^*. A *language* over V is a subset $L \subseteq V^*$. For any languages L and L', we define their concatenation as $LL' = \{ww' \,|\, w \in L, w' \in L'\}$. For any language L, we put $L^0 = \{e\}$ and $L^i = L^{i-1}L$, where $i \geq 1$. The *closure* of a language L is the set $L^* = \cup_{i \in \omega} L^i$. The *prefix language* of a language L is $L^{\sqsubseteq} = \{x \,|\, x \sqsubseteq w \text{ holds for some } w \in L\}$. We say that the prefix problem is decidable for a language L over an alphabet V, if, for an arbitrary word $w \in V^*$, it is decidable whether $w \in L^{\sqsubseteq}$.

We define the operation *letter reduction*, denoted by lr, over words as follows. For any word $w \in V^*$, there exists a unique decomposition $w = v_1^{n_1} \dots v_k^{n_k}$, where $n_1, \dots, n_k \geq 1$ and $v_1, \dots, v_k \in V$, such that $v_i \neq v_{i+1}$ with $1 \leq i < k$. Then $\mathrm{lr}(w) = v_1 \dots v_k$. Roughly speaking, the lr operation substitutes a sequence of a letter by one. We extend the lr operation for languages as $\mathrm{lr}(L) = \cup_{w \in L} \mathrm{lr}(w)$.

Regular languages. The class *REG* of *regular languages* is the smallest class, which obeys the following rules.

(1) For any alphabet V, if $w \in V^*$ then $\{w\}$ is in *REG*.

(2) If $L, L' \in REG$ then $L \cup L'$, LL' and L^* are in *REG*.

That is, *REG* is the smallest class, which contains every singleton language and closed for the closure, the finite union and the concatenation. Clearly, for any alphabet V, the language V^* is regular.

The characterization of *REG* can be found in a wide range of books and papers concerning automata theory and formal languages (e.g. [HU]). A convenient way to define a regular language is to give its construction according to the above construction rules (1) – (2). This yields an expression, possibly with parentheses, where the members are singleton languages, and the operators are the union, the concatenation and the closure, in growing precedence order. These expressions are called *regular expressions*. For example, the regular expression $(\{a\} \cup \{b\})^*c$ defines the language of sequences of a and b followed by c. For brevity, we often write simply w for a singleton language $\{w\}$ in regular expressions in the sequel. Thus we have $(a \cup b)^*c$ for the above expression. Note that a regular language can be defined by several regular expressions.

Finite automata. A *deterministic finite automaton* (DFA) is a 5-tuple $A = (Q, V, \tau, q_0, F)$, where Q is the finite nonempty set of *states*, V is the *input alphabet*, $\tau : Q \times V \to Q$ is the total *transition function*, $q_0 \in Q$ is the *initial state* and $F \subseteq Q$ is the set of *final states*. We extend τ for $Q \times V^*$ with $\tau(q, e) = q$ and $\tau(q, wv) = \tau(\tau(q, w), v)$, where $q \in Q$, $w \in V^*$ and $v \in V$. A state $q \in Q$ is called *trap state* if there exists no word $w \in V^*$ such that $\tau(q, w) \in F$. We assume every state to be *accessible*, that is, for each $q \in Q$, there exists a word $w \in V^*$ such that $\tau(q_0, w) = q$.

A DFA can be represented as a directed labeled graph, where the vertices are the states and the edges are the transitions labeled by the corresponding letters of V. It should be clear that, for any $q, q' \in Q$ and $w \in V^*$, $\tau(q, w) = q'$ implies that there is a path from q to q' labeled by w. By the graph representation, it is easy to show that, for any state $q \in Q$, it can be decided in $O(|Q||V|)$ time, whether q is a trap state. Hence, the subset $T \subseteq Q$ of trap states can be determined in $O(|Q|^2|V|)$ time.

We say that the DFA $A = (Q, V, \tau, q_0, F)$ *recognizes* the word $w \in V^*$ if $\tau(q_0, w) \in F$. The recognizability of any word $w \in V^*$ by A can be decided in $O(|w|)$ time. The language recognized by A is the set $L_A = \{w \in V^* \mid \tau(q_0, w) \in F\}$. We say that the language L is *recognizable* if there exists a DFA A such that $L = L_A$. The following proposition, known as *Kleene's theorem*, is a fundamental result in the theory of automata.

Propositon 2.1 *A language is regular if and only if it is recognizable.*

Moreover, given a regular expression E, a DFA recognizing the language defined by E can be constructed effectively. Conversely, given a DFA A, a regular expression defining L_A can be constructed effectively, too (see [HU]).

Prefixes. The prefix words and languages play very important role in this paper, hence we pay more attention to them. Let $A = (Q, V, \tau, q_0, F)$ be a DFA. Observe that, for any word $w \in V^*$, $w \in L_A^{\sqsubseteq}$ holds if and only if $\tau(q_0, w)$ is not a trap state. Recall that the set of trap states $T \subseteq Q$ can be determined in $O(|Q|^2|V|)$ time. Define the DFA $B = (Q, V, \tau, q_0, Q - T)$, then it should be clear that $L_B = L_A^{\sqsubseteq}$. By Proposition 2.1, we have that if L is a regular language, then $L^{\sqsubseteq}$ is also regular. Hence, the prefix problem is decidable for regular languages.

2.2 The language class $ONREG_0$

We define a subclass $ONREG_0$ of REG. The name $ONREG_0$ refers to certain properties concerning server problems, which are explained later. The class $ONREG_0$ is investigated for the first time in the present paper, hence, in addition to the definition, it is necessary to characterize it. We do this by presenting three different definitions for $ONREG_0$ and proving their equivalence. The first definition shows the inclusion $ONREG_0 \subset REG$ immediately.

Definition 2.2 *The class $ONREG_0$ is the smallest one, which obeys the following rules.*

(1) *For any alphabet V, if $w \in V^*$ then $\{w\}$ and $\{w\}^*$ are in $ONREG_0$.*
(2) *If $L, L' \in ONREG_0$ then LL' and $L \cup L'$ are in $ONREG_0$.*

That is, $ONREG_0$ is the smallest class, which contains every singleton language, the closure of each singleton language, and closed for the finite union and the concatenation. Roughly speaking, a language $L \in ONREG_0$ can be constructed on the same way as a regular one, with the constraint that the closure operation is allowed only for singleton languages. It can be shown that, if V is an alphabet and $|V| > 1$, then $V^* \notin ONREG_0$. The second definition is very useful to prove the main result of the paper.

Definition 2.3 *$ONREG_0$ is the smallest class, which satisfies the following conditions.*

(1) *For any alphabet V and $w \in V^*$, $\{w\}, \{w\}^* \in ONREG_0$.*
(2) *For any alphabet V, $L \in ONREG_0$ and $w \in V^*$, $L\{w\}, L\{w\}^* \in ONREG_0$.*
(3) *If $L, L' \in ONREG_0$ then $L \cup L' \in ONREG_0$.*

The third definition describes explicitly, what kind of languages belongs to $ONREG_0$.

Definition 2.4 *Let V be an arbitrary alphabet and let L be a language over V. Then $L \in ONREG_0$ holds if and only if L can be defined by a regular expressions of the following form. There exist integers $r, s \geq 0$ and words $x_{i,j}, y_{i,j} \in V^*$, where $1 \leq i \leq r$ and $1 \leq j \leq s$, such that*

$$L = \cup_{1 \leq i \leq r} x_{i,1}(y_{i,1})^* \dots x_{i,j}(y_{i,j})^* \dots x_{i,s}(y_{i,s})^*.$$

Finally, we prove that the three definitions are equivalent. We perform this showing that Definition 2.4 is equivalent to both the definitions 2.2 and 2.3. Suppose that a language L can be defined by a regular expression of the form as in Definition 2.4. Then it is easy to see that L can be constructed by the rules of either Definition 2.2 or Definition 2.3. On the other hand, it is a routine exercise to show by structural induction on the rules that any language obeying the rules of either Definition 2.2 or Definition 2.3 can be defined by a regular expression of the form as in Definition 2.4.

2.3 Server problems and satisfying algorithms

Let $M = (V, \delta)$ be a finite metric space, where $V = \{v_1, \dots, v_n\}$ is the set of points and $\delta : V \times V \to \mathbf{R}^+$ is the distance function. Thus, for any $u, v, v' \in V$, $\delta(u, v) = \delta(v, u)$ and $\delta(u, v) \leq \delta(u, v') + \delta(v', v)$ hold. Moreover, $\delta(u, v) = 0$ if and only if u and v are identical. Note that M can be represented as an n-vertex complete graph G_M, where the vertices are labeled by the elements of V and the edges are weighted as determined by δ. We put $\delta_{max} = \max_{1 \leq i,j \leq n} \delta(v_i, v_j)$.

There are k mobile *servers* occupy exactly k vertices of G_M. We assume that $1 < k < n$ and $n \geq 3$. Note that the other cases are trivial and irrelevant from the point of view of our paper (see later). Suppose that there is a server on the vertex v_i and there is no one on v_j. Then moving the server from v_i to v_j costs $\delta(v_i, v_j)$. Let the metric space and the number of servers be arbitrary, but fixed in the sequel. We use the symbols n and k to denote the number of vertices and the number of servers, respectively.

A *configuration* (or a *state*) σ is a word $i_1 \dots i_k \in \{1, \dots, n\}^k$, where $i_j < i_{j+1}$ holds for each $1 \leq j < k$, showing that the servers are on the vertices $v_{i_1}, \dots, v_{i_k}$. It is easy to see that there are exactly $\binom{n}{k}$ different configurations. A configuration can be changed by moving a server from an occupied vertex to an empty one.

A *request* is a letter $v_i \in V$. A *satisfaction* of the request in a given *starting configuration* is a sequence of movements of some servers, such that the resulting configuration contains i, that is, there is a server on the vertex v_i. The cost of the satisfaction is the sum of the costs of its movements. Observe that a request v_i can be satisfied with no movements if and only if the starting configuration contains i. Moreover, any request can be satisfied by one movement of one server in any starting configuration.

A *request word* is a sequence of requests, that is a word over V. A satisfaction of a request word $w = v_1 \dots v_l$ in a given starting configuration is the sequence of the satisfactions of the requests $v_1, \dots, v_l$ after each other. The cost of a satisfaction is the sum of the costs of the satisfactions of composing requests.

Satisfactions are denoted by the symbol S in the sequel. By $|S|$ we mean the cost of the satisfaction S. For any satisfaction S, we denote the starting and the resulting configuration of S by σ_S and $\overline{\sigma}_S$, respectively. A *decomposition of the satisfaction* S of a word w is a sequence $S_1, \dots, S_l$ of satisfactions of words $w_1, \dots, w_l$, such that $w = w_1 \dots w_l$, $\sigma_{S_1} = \sigma_S$ and $\overline{\sigma}_{S_i} = \sigma_{S_{i+1}}$ $(1 \leq i < l)$ hold, and the satisfaction of w by applying $S_1, \dots, S_l$ after each other gives exactly S. Then we write $S = S_1 \dots S_l$.

Clearly, for any request word w, there is a satisfaction with minimal cost in a given starting configuration σ, called an *optimal satisfaction* of w. Denote that cost by $\text{opt}_\sigma(w)$. Let σ and σ' be different configuration, then it should be obvious that $|\text{opt}_\sigma(w) - \text{opt}_{\sigma'}(w)| \leq k\delta_{max}$. If the starting configuration σ is understood, then we write simply $\text{opt}(w)$. Let the starting configuration be arbitrary, but fixed in the sequel.

Let $w \in V^*$ be a word such that $w = w_1 v^n w_2$ holds, for some $w_1, w_2 \in V^*$, $v \in V$ and $n \geq 1$. Then it is easy to show that $\text{opt}(w_1 v^n w_2) = \text{opt}(w_1 v w_2)$. It follows that, for any word w, $\text{opt}(w) = \text{opt}(\text{lr}(w))$ holds.

The following results characterize the word composition and decomposition from the point of view of optimal satisfactions.

Lemma 2.5 *Let $w \in V^*$. Consider an arbitrary decomposition $w = w_1 w_2$, where $w_1, w_2 \in V^*$. Then the following statements hold.*

(1) $\mathit{opt}(w_1) + \mathit{opt}(w_2) \leq \mathit{opt}(w) + k\delta_{max}$

(2) $\mathit{opt}(w) \leq \mathit{opt}(w_1) + \mathit{opt}(w_2) + k\delta_{max}$

Proof. There exist satisfactions, denoted by S, S_1 and S_2, with the costs $\text{opt}(w)$, $\text{opt}(w_1)$ and $\text{opt}(w_2)$ for the request words w, w_1 and w_2, respectively, in the starting configuration σ.

The satisfaction S can be decomposed as $S = S'S''$, where S' is a satisfaction of w_1 in σ, and S'' is a satisfaction of w_2 in $\overline{\sigma}_{S'}$. Observe that it is easy to transform S'' to a satisfaction of w_2 in σ such that, before starting S'', we change the starting configuration σ to $\overline{\sigma}_{S'}$ by moving the appropriate servers. Clearly, this modification costs at most $k\delta_{max}$. Now suppose the contrary of (1), that is, $\text{opt}(w_1) + \text{opt}(w_2) > \text{opt}(w) + k\delta_{max}$. Then, by the above results, it is easy to see that either S' must be a more optimal satisfaction of w_1 than S_1 in σ, or S'' with the above modification should cost less than S_2 in σ. Both contradict that S_1 and S_2 have optimal costs, hence the statement (1) holds.

Now consider the following satisfaction of w in σ. Satisfy the prefix w_1 by S_1, then recover the starting configuration σ and satisfy the suffix w_2 by S_2. The satisfaction S_1 costs $\text{opt}(w_1)$, the recovery of the starting configuration costs at most $k\delta_{max}$ and S_2 costs $\text{opt}(w_2)$. Hence, the cost of the satisfaction is at most $\text{opt}(w_1) + \text{opt}(w_2) + k\delta_{max}$, which implies (2). □

Roughly speaking, for a given metric space and number of servers, our goal is to find satisfactions with minimal costs for request words. More precisely, we want to find an algorithm, which gives server moving sequences with minimal cost to satisfy any possible request word.

In this paper we consider such kind of algorithms, which are deterministic and computes the satisfaction of any request word in any starting configuration, that is, the server moving sequence of the satisfactions letter by letter. For the sort, we call simply *algorithm* the ones in the sequel, which have the above properties.

Let $\mathcal{A}$ be an algorithm, let w be a request word and let σ be the starting configuration. Then the cost of the satisfaction of w given by $\mathcal{A}$ in σ is denoted by $\mathcal{A}_\sigma(w)$. If σ is understood, we write simply $\mathcal{A}(w)$.

We say that an algorithm is *lazy*, if, for any request and starting configuration, it moves at most one server to satisfy the request. Otherwise, the algorithm is said *eager*. Clearly, for any request word, there exists a lazy algorithm which satisfies it. The following statement has been shown in [MMS].

Propositon 2.6 *For every eager algorithm, there exists a lazy one such that, for any request word, the satisfaction given by the lazy one costs no more than the satisfaction given by the eager one.*

This result provides that it is enough to find an eager algorithm, if we want to show the existence of a lazy one with respect to any cost limit. If the type of an algorithm is not defined explicitly, we mean eager one in the sequel.

There is an another classification of algorithms. An algorithm is said *off-line*, if it reads the whole request word first, then computes a satisfaction of that one. Moreover, an algorithm is called *on-line*, if it reads the request words from left to right, and reading a letter it gives the satisfaction of that request before reading the next one.

Let $c \geq 1$ be a real number and let $L \subseteq V^*$ be a language. An algorithm $\mathcal{A}$ is said *c-competitive* on L (with the constant K), if $\mathcal{A}(w) \leq c\,\mathrm{opt}(w) + K$ holds for each $w \in L$, where K depends on only $\mathcal{A}$ and L. Clearly, if an algorithm $\mathcal{A}$ is c-competitive on a language L, then it is c-competitive on any sublanguage $L' \subseteq L$, too. Hence, if an algorithm is efficient on V^*, then it is efficient on any language over V. Obviously, for any alphabet V, there exists a 1-competitive off-line algorithm on V^*.

We note that it has been proved in [CKPV] that there exists an off-line algorithm, which gives the optimal satisfaction of any word $w \in V^*$ in $O(k|w|^2)$ time. Hence, we can conclude that, for an arbitrary language L, there exists a polynomial 1-competitive off-line algorithm on L.

The class of on-line algorithms has not so nice property. Up to this time, the known best competitive ratio of any on-line algorithm on V^* is $2k-1$ (see [KP]). Moreover, a lower bound was presented in [MMS], which shows that the competitive ratio of any on-line algorithm on V^* is at least k. The question naturally arises that is it possible to reduce the competitive factor of on-line algorithms, if we consider only a special class of languages (e.g. $ONREG_0$)? Recall that $V^* \in REG$ holds, for any alphabet V, hence REG is not suitable for this purpose.

If it is not defined explicitly, by an algorithm we mean an on-line one in the rest of the paper.

A language $L \subseteq V^*$ is said *c-competitive*, if, for every M and k, there exists an algorithm, which is c-competitive on it. The 1-competitive languages are called *on-line languages*. The class of all on-line languages is denoted by $ONLINE$.

Lemma 2.7 *Every finite language is on-line.*

Proof. Suppose $L = \{w_1, \dots, w_m\}$ is a finite language. We put $l = \max_{1 \leq i \leq m} |w_i|$. Let $\mathcal{A}$ be an arbitrary algorithm for L. Clearly, for any $w \in L$, $\mathcal{A}(w) \leq opt(w) + l\delta_{max}$. □

The properties of prefix words and prefix languages concerning the server problem are cornerstones in our proofs. Let L be an on-line language and let the algorithm $\mathcal{A}$ be 1-competitive on L. Suppose that $w \in L$ and $w = w_1 w_2$. Then, by (2) of Lemma 2.5, $\mathcal{A}(w) = \mathcal{A}(w_1 w_2) \leq \text{opt}(w) + K \leq \text{opt}(w_1) + \text{opt}(w_2) + k\delta_{max} + K$. Now let $S = S_1 S_2$ be the satisfaction of w given by $\mathcal{A}$, where S_1 is the satisfaction of the prefix w_1 and S_2 is the satisfaction of the suffix w_2. Clearly, $|S_1| \geq \text{opt}(w_1)$ and $k\delta_{max} + |S_2| \geq \text{opt}(w_2)$. Hence the following statements hold.

Observation 2.8 *If $\mathcal{A}$ is a 1-competitive algorithm on a language L with the constant K, $w \in L$ and $w = w_1 w_2$, then*
(1) $\mathcal{A}(w_1) \leq opt(w_1) + 2k\delta_{max} + K$ *and*
(2) *the cost of $\mathcal{A}$ on the suffix w_2 is no more than $opt(w_2) + k\delta_{max} + K$.*

By (1) of Observation 2.8, we have the following result immediately.

Theorem 2.9 *If $\mathcal{A}$ is a 1-competitive algorithm on a language L with the constant K, then $\mathcal{A}$ is 1-competitive on $L^{\sqsubseteq}$ with the constant $2k\delta_{max} + K$.*

3 The languages in $ONREG_0$ are on-line

The name $ONREG_0$ refers to the property of this class that it consists of on-line regular languages. However, it does not contain all on-line regular language.

In this section we prove that the languages in $ONREG_0$ are on-line. We do this by executing structural induction on the rules of Definition 2.3.

As the basic step of the structural induction, we prove that the languages $\{w\}$ and $\{w\}^*$ are on-line, for any word w (see (1) of Definition 2.3).

Lemma 3.1 *Let w be an arbitrary word over an arbitrary alphabet. Then $\{w\}$ and $\{w\}^*$ are on-line languages.*

Proof. As for the on-lineness of the language $\{w\}$, it immediately follows from Lemma 2.7. However, it also implied by the on-lineness of $\{w\}^*$, since $\{w\} \subseteq \{w\}^*$.

We construct a 1-competitive algorithm $\mathcal{A}$ on $\{w\}^*$. Informally speaking, $\mathcal{A}$ works as follows. We find a satisfaction $\underline{S}$ on an appropriate w-sequence, which has minimal cost density and results the same configuration as the starting one. Then $\mathcal{A}$ applies $\underline{S}$ repeatedly on any w-sequence.

We need some preparations. A satisfaction S of a request word w^{m_S} $(m_S \geq 1)$ is called *circular* on w, if $\sigma_S = \overline{\sigma}_S$ holds. Denote the set of all circular satisfactions on w by $CSAT_w$. Moreover, a satisfaction S of a word w^{m_S} $(m_S \geq 0)$ is called *noncircular* on w, if either $m_S = 0$, or $m_S \geq 1$ and $S = S_1 \dots S_{m_S}$, where each S_i is a satisfaction of the word w and $\sigma_{S_i} \neq \overline{\sigma}_{S_j}$, for any $1 \leq i \leq j \leq l$. Denote the set of all noncircular satisfactions on w by $NCSAT_w$. Observe that, since the number of different configurations is $\binom{n}{k}$, $m_S < \binom{n}{k}$ should hold for any noncircular satisfaction S.

Let the satisfaction $\underline{S} \in CSAT_w$ be such that $\frac{|\underline{S}|}{m_{\underline{S}}}$ is minimal in $CSAT_w$ and $m_{\underline{S}}$ is minimal with respect to the subset of $CSAT_w$ determined by the previous condition. Roughly speaking, $\underline{S}$ is one of the shortest satisfactions in $CSAT_w$, which has the minimal cost density.

We show that $\underline{S}$ exists and it can be computed. We prove this by showing that it is enough to consider only such kind of satisfactions S, for which $m_S \leq \binom{n}{k} - 1$ holds. Suppose that $m_{\underline{S}} > \binom{n}{k} - 1$. Decompose $\underline{S}$ as $\underline{S} = S_1 \dots S_{m_{\underline{S}}}$, where each S_i is a satisfaction of the word w, for any $1 \leq i \leq m_{\underline{S}}$. Clearly, $\sigma_{\underline{S}} = \sigma_{S_1} = \overline{\sigma}_{\underline{S}} = \overline{\sigma}_{S_{m_{\underline{S}}}}$ and $\overline{\sigma}_{S_i} = \sigma_{S_{i+1}}$. The number of different configurations is $\binom{n}{k}$, hence at least one of the following cases holds.

(i) There is an integer i, where $1 < i \leq m_{\underline{S}}$, such that $\sigma_{S_i} = \sigma_{\underline{S}}$. Let $S_1' = S_1 \dots S_{i-1}$ and $S_2' = S_i \dots S_{m_{\underline{S}}}$. Then $\underline{S} = S_1'S_2'$ and $S_1', S_2' \in CSAT_w$ hold. Clearly, $m_{\underline{S}} = m_{S_1'} + m_{S_2'}$ and $|\underline{S}| = |S_1'| + |S_2'|$. Hence, either $\frac{|S_1'|}{m_{S_1'}} \leq \frac{|\underline{S}|}{m_{\underline{S}}}$ or $\frac{|S_2'|}{m_{S_2'}} \leq \frac{|\underline{S}|}{m_{\underline{S}}}$, which contradicts the minimality of $m_{\underline{S}}$.

(ii) There are integers i and j, where $1 < i < j < m_{\underline{S}}$, such that $\sigma_{S_i} = \sigma_{S_j}$. Let $S_1' = S_1 \dots S_{i-1}$, $S_2' = S_i \dots S_{j-1}$ and $S_3' = S_j \dots S_{m_{\underline{S}}}$. Then $\underline{S} = S_1'S_2'S_3'$ and $S_1'S_3', S_2' \in CSAT_w$ hold. Clearly, $m_{\underline{S}} = m_{S_1'S_3'} + m_{S_2'}$ and $|\underline{S}| = |S_1'S_3'| + |S_2'|$. It is easy to see that either $\frac{|S_1'S_3'|}{m_{S_1'S_3'}} \leq \frac{|\underline{S}|}{m_{\underline{S}}}$ or $\frac{|S_2'|}{m_{S_2'}} \leq \frac{|\underline{S}|}{m_{\underline{S}}}$, which contradicts the minimality of $m_{\underline{S}}$.

We have $m_{\underline{S}} \leq \binom{n}{k} - 1$. The subset of $CSAT_w$, which consists of the satisfactions S obeying $m_S \leq \binom{n}{k} - 1$, is finite, hence $\underline{S}$ can be computed.

Let now the algorithm $\mathcal{A}$ work as follows. Starting the satisfaction of a request word in $\{w\}^*$, $\mathcal{A}$ changes the starting configuration to $\sigma_{\underline{S}}$, then applies the satisfaction $\underline{S}$ repeatedly to the end of the input word. Clearly, $\mathcal{A}$ is on-line on $\{w\}^*$.

We prove that $\mathcal{A}$ is 1-competitive on $\{w\}^*$. Suppose that the input word is w^m, where $m \geq 0$. Let p the smallest integer such that $p \geq \frac{m}{m_{\underline{S}}}$. Then, by the definition of $\mathcal{A}$,

$$\mathcal{A}(w^m) \leq k\delta_{max} + p|\underline{S}| \qquad (*)$$

holds. Denote by S an optimal satisfaction of w^m in the given starting configuration, thus $|S| = opt(w^m)$. It is an easy exercise to show that the satisfaction

S can be decomposed as $S = S_0\hat{S}_1S_1\ldots\hat{S}_lS_l$, where $\hat{S}_1, \hat{S}_2, \ldots, \hat{S}_l \in CSAT_w$ and $S_0S_1\ldots S_l \in NCSAT_w$. By the property of noncircular satisfactions, we have $m_{S_0\ldots S_l} < \binom{n}{k}$. Moreover, by the choice of $\underline{S}$, $\frac{|\underline{S}|}{m_{\underline{S}}} \leq \frac{|\hat{S}_i|}{m_{\hat{S}_i}}$ holds, for each $1 \leq i \leq l$. Now we can calculate as follows.

$$\begin{aligned}\mathcal{A}(w^m) - \mathrm{opt}(w^m) &\leq p|\underline{S}| + k\delta_{max} - \textstyle\sum_{1\leq i\leq l} |\hat{S}_i| \\ &\quad \text{(by } (*) \text{ and } |S| = \mathrm{opt}(w^m)) \\ &\leq k\delta_{max} + (pm_{\underline{S}} - \textstyle\sum_{1\leq i\leq l} m_{\hat{S}_i})\frac{|\underline{S}|}{m_{\underline{S}}} \\ &\quad \text{(by } \tfrac{|\underline{S}|}{m_{\underline{S}}} \leq \tfrac{|\hat{S}_i|}{m_{\hat{S}_i}}).\end{aligned}$$

Moreover, since

$$\begin{aligned}pm_{\underline{S}} - \textstyle\sum_{1\leq i\leq l} m_{\hat{S}_i} &\leq pm_{\underline{S}} - m + \tbinom{n}{k} \\ &\quad \text{(by } m = \textstyle\sum_{1\leq i\leq l} m_{\hat{S}_i} + m_{S_0\ldots S_l},\ m_{S_0\ldots S_l} < \tbinom{n}{k}) \\ &\leq m_{\underline{S}} + \tbinom{n}{k} \\ &\quad \text{(by the choice of } p),\end{aligned}$$

we have

$$\mathcal{A}(w^m) \leq \mathrm{opt}(w^m) + (k\delta_{max} + |\underline{S}| + \frac{|\underline{S}|}{m_{\underline{S}}}\binom{n}{k}).$$

Hence $\mathcal{A}$ is 1-competitive on $\{w\}^*$. □

In the next step of the structural induction, we prove that the two construction rules defined in (2) of Definition 2.3 preserve the on-lineness.

Lemma 3.2 *For any alphabet V, language $L \in ONREG_0$ and word $w \in V^*$, if L is on-line, then $L\{w\}$ and $L\{w\}^*$ are also on-line languages.*

Proof. Since $L\{w\} \subseteq L\{w\}^*$ holds, it is enough to show that $L\{w\}^*$ is on-line to prove the lemma.

The language L is on-line, hence there is an algorithm $\mathcal{A}_1$, which is 1-competitive on it. Moreover, by Lemma 3.1, the language $\{w\}^*$ is on-line. Let $\mathcal{A}_2$ be a 1-competitive algorithm on $\{w\}^*$. We construct a 1-competitive algorithm $\mathcal{A}$ on $L\{w\}^*$. Informally, $\mathcal{A}$ applies $\mathcal{A}_1$ as long as possible, then finds the beginning of the remainder w-sequence (if there is) and applies $\mathcal{A}_2$ to the end of the request word.

Observe that every input word is of the form xw^m, where $x \in L$ and $m \geq 0$. Let $x \in L$ and $m \geq 1$ be arbitrary, but fixed in the sequel. (As for the case $m = 0$, the 1-competitiveness of $\mathcal{A}$ will follow from its construction immediately.) Observe that xw^m can be decomposed unambiguously as $xw^{m_1}w_1w_2w^{m_2}$, where $m = m_1 + m_2 + 1$, $w_1w_2 = w$ and $xw^{m_1}w_1$ is the longest prefix of xw^m, which is in $L^{\sqsubseteq}$.

Let the algorithm $\mathcal{A}$ work as follows.

1. While the scanned prefix of the input word xw^m is in $L^{\sqsubseteq}$, $\mathcal{A}$ applies $\mathcal{A}_1$. Observe that in this way $\mathcal{A}_1$ is applied exactly on $xw^{m_1}w_1$.
2. Then $\mathcal{A}$ reads the letters successing $xw^{m_1}w_1$, satisfies them arbitrarily and stores their concatenation, until the stored word is of the form yw or the

input ends. Since $|w_2| \leq |w|$, it should be clear that, for any input word, less than $2|w|$ letters are to be read in this step.

3. If there are no more letters left on the input, then $\mathcal{A}$ terminates. Otherwise, observe that $w_2 w^{m_2} = yww^{m_2-1}y'$ should hold, for some y, where $yy' = w_2$ and $y' \sqsubseteq w$. Moreover, the rest of the input is $w^{m_2-1}y'$, hence $\mathcal{A}_2$ can be applied on the remainder input without any difficulties. Let $\mathcal{A}$ apply $\mathcal{A}_2$ on the rest of the input.

Now we prove that $\mathcal{A}$ is 1-competitive on $L\{w\}^*$. Supposing $xw^m \in L^{\sqsubseteq}$ $(m \geq 0)$, by Lemma 2.9, $\mathcal{A}(xw^m) \leq \text{opt}(xw^n) + K_1 + 2k\delta_{max}$ holds. Now suppose that $xw^m \notin L^{\sqsubseteq}$. Clearly, in this case $m \geq 1$ should hold. We can calculate as follows.

$$\begin{aligned}
\mathcal{A}(xw^n) &\leq \mathcal{A}_1(xw^{m_1}w_1) + 2|w|\delta_{max} + \mathcal{A}_2(w^{m_2}) \\
&\quad \text{(by the construction of } \mathcal{A}) \\
&\leq \text{opt}(xw^{m_1}w_1) + K_1 + 2k\delta_{max} + \text{opt}(w^{m_2}) + K_2 + 2|w|\delta_{max} \\
&\quad \text{(by Lemma 2.9)} \\
&\leq \text{opt}(xw^{m_1}w_1w_2) + \text{opt}(w^{m_2}) + K_1 + K_2 + 2(k + |w|)\delta_{max} \\
&\leq \text{opt}(xw^{m_1}w_1w_2w^{m_2}) + k\delta_{max} + K_1 + K_2 + 2(k + |w|)\delta_{max} \\
&\quad \text{(by Lemma 2.5)} \\
&= \text{opt}(xw^m) + (K_1 + K_2 + (3k + 2|w|)\delta_{max})
\end{aligned}$$

Hence $\mathcal{A}$ is 1-competitive on $L\{w\}^*$. □

Finally, we have to check the construction rule defined in (3) of Definition 2.3 to complete the proof.

Lemma 3.3 *For any languages $L_1, L_2 \in ONREG_0$, if both L_1 and L_2 are on-line then the language $L_1 \cup L_2$ is on-line, too.*

Proof. Since L_1 and L_2 are on-line languages, there exist algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ such that, for any words $w_1 \in L_1$ and $w_2 \in L_2$, $\mathcal{A}_1(w_1) \leq \text{opt}(w_1) + K_1$ and $\mathcal{A}_2(w_2) \leq \text{opt}(w_2) + K_2$ hold, where K_1 and K_2 are constants. We construct an algorithm $\mathcal{A}$, which is 1-competitive on $L_1 \cup L_2$.

Let the input word be an arbitrary $w \in L_1 \cup L_2$. The algorithm $\mathcal{A}$ works as follows.

1. Reading the next letter of w, on the basis of the scanned prefix $w' \sqsubseteq w$, $\mathcal{A}$ tries to decide that w which of the languages L_1 and L_2 belongs to. Recall that both $w' \in L_1^{\sqsubseteq}$ or $w' \in L_2^{\sqsubseteq}$ are decidable. While this cannot be decided unambiguously, $\mathcal{A}$ computes the satisfactions and the new configurations determined by both $\mathcal{A}_1$ and $\mathcal{A}_2$, but satisfies the request as suggested by $\mathcal{A}_1$. However, it stores the configuration computed by $\mathcal{A}_2$.

2. If $\mathcal{A}$ detects that w cannot be in L_2, then it finishes the satisfaction of w continuing by $\mathcal{A}_1$. Otherwise, if it turns out that w is not in L_1, then $\mathcal{A}$ changes the configuration to the one stored for $\mathcal{A}_2$, and completes the satisfaction of w by $\mathcal{A}_2$.

We show that it is 1-competitive on $L_1 \cup L_2$. Observe that, for any input word $w \in L_1 \cup L_2$, exactly one of the following two cases holds.

(i) $w \in L_1^{\sqsubseteq}$. By the construction of $\mathcal{A}$, in this case $\mathcal{A}(w) = \mathcal{A}_1(w)$ holds. Since $\mathcal{A}_1$ is 1-competitive on L_1, by Proposition 2.9 we have $\mathcal{A}(w) \leq \text{opt}(w) + K_1 + 2k\delta_{max}$, where K_1' is a constant.

(ii) $w \in (L_2 - L_1^{\sqsubseteq})$. In this case $\mathcal{A}$ works as follows. The word w can be decomposed unambiguously as $w = w_1 w_2$, where w_1 is the longest prefix of w such that $w_1 \in L_1^{\sqsubseteq}$. The algorithm $\mathcal{A}$ implements $\mathcal{A}_1$ on w_1, changes the configuration and implements $\mathcal{A}_2$ on w_2. Recall that changing the configuration costs at most $k\delta_{max}$. We can calculate as follows.

$$\begin{aligned}
\mathcal{A}(w) &\leq \mathcal{A}_1(w_1) + k\delta_{max} + \text{opt}(w_2) + K_2 + k\delta_{max} \\
&\quad \text{(by (2) of Observation 2.8)} \\
&\leq \text{opt}(w_1) + K_1 + 2k\delta_{max} + \text{opt}(w_2) + K_2 + 2k\delta_{max} \\
&\quad \text{(by Lemma 2.9)} \\
&\leq \text{opt}(w) + (K_1 + K_2 + 4k\delta_{max}) \\
&\quad \text{(by (1) of Lemma 2.5)}
\end{aligned}$$

We have that $\mathcal{A}$ is 1-competitive on $L_1 \cup L_2$. □

With this, we are ready to prove the main result of our paper.

Theorem 3.4 *Every language in $ONREG_0$ is on-line. Moreover, given a language $L \in ONREG_0$, an algorithm can be constructed effectively, which is 1-competitive on L.*

Proof. Recall that every language $L \in ONREG_0$ can be constructed as described in Definition 2.3. Hence, by the lemmas 3.1, 3.2, 3.3 and by the principline of structural induction, we have that $L \in ONLINE$ holds. Moreover, by the application of the constructions in the proofs of the above lemmas, a 1-competitive algorithm can be given for L. □

Recall that $\text{opt}(w) = \text{opt}(\text{lr}(w))$ holds, for any word w. Moreover, the competitive ratio of any algorithm on any sublanguage of a given language is no more, than its competitive ratio on the whole language. By these observations, we can show the on-lineness of certain other languages, which can be even not regular.

Corollary 3.5 *For an arbitrary language L, if there exists a language $L' \in ONREG_0$ such that $L \subseteq L'$ or $lr(L) \subseteq L'$ hold, then L is on-line. Moreover, in this case a 1-competitive algorithm can be constructed effectively for L.*

Finally, we present three examples for the application of Corollary 3.5.

(1) The language $L_1 = \{a^n b^n c^n \mid n \geq 0\}$ is a well known nonregular language. However, by Corollary 3.5, it is on-line, since $\text{lr}(L_1) = \{e, abc\} \in ONREG_0$.

(2) Consider $L_2 = \{w_1^n w_2^n \mid n \geq 0\}$, where w_1 and w_2 are arbitrary words. Note that generally L_2 is not regular. However, $L_2 \in ONLINE$ follows from the fact that L_2 is a subset of the language $(w_1)^*(w_2)^*$, which is in $ONREG_0$.

(3) It can be shown that $L_3 = (a \cup abc)^*$ is not in $ONREG_0$, but it is regular. Observe that L_3 can be defined by $(a^*abc)^*a^*$ as well. Since $\mathrm{lr}((a^*abc)^*a^*) \subseteq (abc)^*a^* \in ONREG_0$, we have that L_3 is on-line language.

4 Related results

The question obviously arises that if there exists an on-line language L, of which the on-lineness cannot be proved by Corollary 3.5? (That is, such L that $\mathrm{lr}(L)$ is not a subset of any language in $ONREG_0$.) Specially, is there such kind of language in REG? These problems are open up to this time. However, we have the following results, which shows that if there exists a language in REG with the above property, then it should be very special one.

Recall that the construction of a language in $ONREG_0$ differs from the construction of a general regular one in the point that the closure operation is allowed only for singleton languages. Moreover, by Theorem 3.4, every language in $ONREG_0$ is necessarily on-line. Hence one can guess that, roughly speaking, a regular language may loose the on-lineness, when the closure is applied for a multielement set during its construction. The following lemma shows that this really holds in most cases.

Theorem 4.1 *Let V be an arbitrary alphabet with $|V| \geq 3$. Consider any two words $w_1, w_2 \in V^*$ such that w_1 contains at least two different letters and there is a letter in w_2, which does not occur in w_1. Then the language $(w_1 \cup w_2)^*$ is not on-line.*

Proof. Assume that the number of servers (i.e. k) is 2. It is sufficient to show that there exists a metric space $M = (V, \delta)$, in which the competitive ratio of L is greater than 1.

Denote by a the letter, which occurs in w_2 and not contained by w_1. For any different letters $u, v \in V - \{a\}$, let $\delta(u, v) = 1$ and, for each $v \in V - \{a\}$, let $\delta(v, a) = D$, where D is defined later. Suppose that the starting configuration always contains a server on a.

Let us consider a request-answer game, where a requester R plays against an algorithm $\mathcal{A}$. In each round, R gives a request, which is satisfied by $\mathcal{A}$ immediately. We show that defining an appropriately large D, for any algorithm $\mathcal{A}$, R has a strategy providing that the difference of the costs of the satisfaction generated by $\mathcal{A}$ and the optimal one grows unboundedly. This implies that the competitive ratio of L is greater than 1.

Let $H = \{w_2w_1w_2, \ldots, w_2w_1^tw_2\}$, where $t > 1$ is defined later. We assume that the request words composed by R are chosen from H^*. Since $H^* \subseteq L$, to prove the lemma, it is enough to show that H^* is not on-line. Observe that any word $w \in H^*$ consists of sections of the form $w_2 w_1^s w_2$, where $1 \le s \le t$.

Let R compose its request words dynamically, sections by sections, obeying the following rules.

Rule 1 If, for some $1 \le s < t$, there is a server on a, when $w_2 w_1^{s-1}$ is satisfied, and $\mathcal{A}$ moves that server processing the next (sth) w_1, then let this section be $w_2 w_1^s w_2$. Specially, if $\mathcal{A}$ moves the server from a while satisfying the initial w_2, let R chose $w_2 w_1 w_2$.

Rule 2 If $\mathcal{A}$ does not move the server from a while processing the w_1-s, then let R choose $w_2 w_1^t w_2$.

We need some technical preparations. Since w_1 contains at least two different letters, it should be clear that $|\mathrm{lr}(w_1)| \ge 2$. Moreover, for any w_1, one of the following cases holds, where $s \ge 1$.

Case 1 $\mathrm{first}(w_1) \ne \mathrm{last}(w_1)$. Then $|\mathrm{lr}(w_1^s)| = s|\mathrm{lr}(w_1)|$.

Case 2 $\mathrm{first}(w_1) = \mathrm{last}(w_1)$. Then $|\mathrm{lr}(w_1^s)| = s(|\mathrm{lr}(w_1)| - 1) + 1$.

Suppose that a section is chosen by Rule 1. Then the cost of the satisfaction by $\mathcal{A}$ is at least $(s-1)|\mathrm{lr}(w_1)| + D$ in Case 1, and $(s-1)(|\mathrm{lr}(w_1)| - 1) + 1 + D$ in Case 2. However, if we do not move the server on a, then this section could be satisfied with cost no more than $|\mathrm{lr}(w_2)| + s|\mathrm{lr}(w_1)| + |\mathrm{lr}(w_2)|$ in Case 1, and $|\mathrm{lr}(w_2)| + s(|\mathrm{lr}(w_1)| - 1) + 1 + |\mathrm{lr}(w_2)|$ in Case 2. Hence, if

$$D > |\mathrm{lr}(w_1)| + 2|\mathrm{lr}(w_2)|$$

holds, then $\mathcal{A}$ is more expensive on this section than the our one.

Now suppose that the section is determined by Rule 2, that is $\mathcal{A}$ does not move the server from a during the processing of w_1-s. Then the cost of $\mathcal{A}$ is at least $t|\mathrm{lr}(w_1)|$ in Case 1, and $t(|\mathrm{lr}(w_1)| - 1) + 1$ in Case 2. However, if we move the server from a to $\mathrm{first}(w_1)$ after the initial w_2, leave there while processing the w_1-s, and move back to a before the final w_2, then this section costs at most $|\mathrm{lr}(w_2)| + D + t(|\mathrm{lr}(w_1)| - 1) + D + |\mathrm{lr}(w_2)|$ in Case 1, and $|\mathrm{lr}(w_2)| + D + t(|\mathrm{lr}(w_1)| - 2) + D + |\mathrm{lr}(w_2)|$ in Case 2. Therefore, if

$$t > 2|\mathrm{lr}(w_2)| + 2D$$

holds, then $\mathcal{A}$ is more expensive than the our one.

For any words w_1 and w_2, the values of D and t can be computed and fixed as above. We have that, for an arbitrary on-line algorithm $\mathcal{A}$, R has a strategy, which proves that $\mathcal{A}$ costs more than an off-line algorithm. Since an input word $w \in H^*$ can contain arbitrary many sections, this difference grows unboundedly, hence $\mathcal{A}$ is not 1-competitive. This implies that H^* (and hence L) is not on-line language. □

Generally, a regular language is defined by a regular expression. The following result shows that if a subexpression defines a not on-line language, then the language defined by the whole expression cannot be on-line.

Theorem 4.2 *Let the language L defined by the regular expression E. Consider any subexpression E' of E. If the language defined by E' is not on-line then L is not on-line, too.*

Proof. It is a routine exercise to prove the theorem using structural induction on the defining rules of REG. $\square$

We show two examples for the application of theorems 4.1 and 4.2.

(1) The language $(a \cup bc)^*$ is not on-line by Theorem 4.1.

(2) Consider the regular expression $((abc \cup bcd)^* d \cup abcd)^*$. Its subexpression $(abc \cup bcd)^*$ defines a not on-line language by Theorem 4.1, hence, by Theorem 4.2, the language defined by the whole expression is not on-line, too.

Finally, we summarize some abstract properties of the language class $ONLINE$ in the following theorem.

Theorem 4.3 *Let L_1 and L_2 be arbitrary on-line languages, then*
(1) for any $L \subseteq L_1$, L is on-line,
(2) $L_1^{\sqsubseteq}$ is on-line,
(3) $L_1 \cap L_2$ is on-line,
(4) if the prefix problem is decidable both for L_1 and L_2, then $L_1 \cup L_2$ is on-line,
(5) L_1^* is generally not on-line,
(6) $\overline{L_1}$ is generally not on-line.

Proof. The statements (1) and (2) have been proved earlier in this paper. Moreover, (3) follows from (1) immediately. We can get (4) by slightly modifying the proof of Lemma 3.3. The statement (5) can be proved by Theorem 4.1. For proving (6), let us assume, that $L_1 = \{w\}^*$, and let u be a letter, which is different from the last letter of w. Let w_1 and w_2 be any words satisfying the conditions in Theorem 4.1. Then we have that $(w_1 u \cup w_2 u)^* \subseteq \overline{L_1}$. Thus, $\overline{L_1}$ is not on-line language. $\square$

References

[BIRS] Borodin, A., Irani, S., Raghavan, P., Schieber, B., *Competitive Paging with Locality of Reference*, STOC 91, pp. 249-259

[CKPV] Chrobak, M., Karloff, H., Payne, T. and Vishwanathan, S., *New Results on Server Problems*, SIAM J. Disc. Math., Vol. 4, No. 2, May 1991, pp. 172-181

[FK] Fiat, A., Karlin, A., *Randomized and Multipointer Paging with Locality of Reference*, STOC 95, pp. 626-634

[KP] Koutsoupias, E. and Papadimitrou, C., *On the k-Server Conjecture*, STOC 94, pp. 507-511

[HU] Hopcroft, J. E. and Ullman, J. D., *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, Series in Computer Science, 1979

[MMS] Manasse, M. S., McGeoch L. A. and Sleator, D. D., *Competitive Algorithms for Server Problems*, Journal of Algorithms 11 (1990), pp. 208-230

[ST] Sleator, D. D., Tarjan, R. E., *Amortized Efficiency of List Update and Paging Rules*, Comm. of the ACM, February 1985, pp. 202-208

[W] Winskel, G., *The Formal Semantics of Programming Languages, An Introduction*, The MIT Press, Foundations of Computing Series, 1993

Received May, 1996