Erasing automata recognize more than context-free languages

František Mráz, Martin Plátek

Abstract. An erasing automaton is a linear bounded Turing machine which can rewrite any symbol on its tape only to a special symbol @. In [1] was formulated a hypothesis that erasing automata cannot recognize all context-free languages (*CFL*). We show here that the opposite is true. We consider erasing automaton (*E*-automaton) as a special type of list automaton studied in [10].

1991 Mathematics Subject Classification: 68Q45, 68Q68

1 Introduction

This paper shows a result contained in the conference contribution [8], and a separation theorem from the conference paper [5]. The first result has started further intensive investigations of erasing and deleting automata resulting in a series of contributions to various conferences and workshops (e.g. [5, 6, 7, 9]), but still in the only one journal paper ([4]). We try to reduce this gap by this paper.

The main result of this paper was shown in a different way also in [5]; nevertheless the technique presented here is interesting on its own and should justify the paper. It is can be viewed as a generalization of a technique used in parallel algorithms for parsing context free languages ([11]).

An erasing automaton is a linear bounded Turing machine which can rewrite any symbol on its tape only to a special symbol @. In [1] there was formulated a hypothesis that erasing automata cannot recognize all context-free languages (*CFL*). We show here that the opposite is true. We consider erasing automaton (*E*-automaton) as a special type of list automaton studied in [10]. The input- and working-memory of a list automaton is a linear doubly linked list with left and right sentinels. Every item of the list contains one symbol from a given alphabet. The automaton has a head, which can read the contents of the visited item. At the beginning of a computation the list contains an input string delimited by two sentinels.

The automaton can perform following basic operations:

MVR - moving to the right,

MVL - moving to the left,

DEL - deleting the visited item and moving the head to its left neighbour, *ER* -rewriting of the contents of the scanned item with a special symbol @, which can not be rewritten again, this operation is referred to as erasing.

Definition 1. A *list automaton* is a system $M = (Q, A, B, I, q_0, F)$, where

- Q is a finite set of *states*,
- A is a finite *input alphabet*,
- B is a finite *working alphabet* containing A, sentinels #,$ and a special symbol @ – all these symbols are not in A,
- q_0 is the *initial state* ($q_0 \in Q$),
- F is a set of *final (accepting) states* (F is a subset of Q),
- I is a finite set of *instructions* of the following form:

$$[q_1, b] \rightarrow [q_2, o],$$

where $q_1, q_2 \in Q$, $b \in B$, o is one of operations *MVR*, *MVL*, *DEL* or *ER*.

Let us consider the instruction mentioned above. The *instruction can be performed*, when M is in the state q_1 and the head is scanning the symbol b. The execution of the instruction means that M changes its state to q_2 and the operation o is performed.

We call M *deterministic* if for any instruction of the form $[q, b] \rightarrow [q_1, o]$, the set I does not contain any other instruction of the form $[q, b] \rightarrow [q_2, o_1]$.

We represent a *configuration* of M by a triple $K = (wb, q, v)$, where w represents the string of symbols in the list on the left side from the head (lefthand side of K), where b means the symbol visited by the head, v is the string in the righthand side part of K and q means the actual state of M.

Let $K_1 = (wb, q, cv)$ be a configuration of M, c, b symbols from B, w, v words from B^*, $i = [q, b] \rightarrow [q_1, o]$ some instruction of M. We write $K_1 \Rightarrow K_2$ *by* i in the following cases:

(a) $o = MVR$, $K_2 = (wbc, q_1, v)$;

(b) $o = DEL$, $K_2 = (w, q_1, cv)$;

(c) $o = MVL$, $K_2 = (w, q_1, bcv)$;

(d) $o = ER$, $K_2 = (w@, q_1, cv)$.

The notation $K_1 \Rightarrow K_2$ means, that there is an instruction i in the previous sense. We say that $K_1 \Rightarrow K_2$ is a *step from* K_1 *to* K_2 . The reflexive and transitive closure of the relation $\Rightarrow$ is denoted by $\Rightarrow^*$.

We say that a configuration of the form $(\#, q_0, v\$)$ is a *starting configuration* of M and a configuration of the form $(\#x\$, q, \lambda)$, where q is from F, is an *accepting* *configuration* (λ denotes the empty word). Any configuration K such that there exists no configuration K_1 such that $K \Rightarrow K_1$ is called a *halting configuration*.

We say that a sequence C of steps $K_1 \Rightarrow K_2$, $K_2 \Rightarrow K_3$, ..., $K_{n-1} \Rightarrow K_n$ is a computation of M. If K_1 is a starting configuration and K_n is an accepting configuration then we call C an *accepting computation*.

We denote $L(M) = \{v \in A^*$; where $(\#, q_0, v\$)$ is a starting configuration of an accepting computation of $M\}$.

Definition 2. A list automaton which uses the operations *MVL*, *MVR*, *ER* only is called *E-automaton*.

A list automaton which uses the operations *MVL*, *MVR*, *DEL* only is called *d-automaton*.

2 The power of erasing automata

It was shown in [10] that for any deterministic context-free language L a deterministic d-automaton M such that $L = L(M)$ can be constructed. The construction of M is based on the idea of the modifications preserving the relation being element of L, i. e. modified word is element of L if and only if the original word is. Such modification cuts off some nonempty part of input word. By execution of a finite number of such modifications we get some short word about which we can decide whether it belongs to L or not. Obviously every d-automaton can be simulated by an E-automaton — simply replace the operation *DEL* by *ER* and add instructions which skip over an erased symbol in the desired direction without changing the current state. Thus the following theorem holds.

Theorem 1. *Any deterministic context-free language can be recognized by a deterministic E-automaton.*

We show a construction of an E-automaton ccepting language given by a context-free grammar. But in the case of nondeterministic languages the method from [10] cannot be used because the modification preserving the relation being element of L cannot be done.

We use a method by which an E-automaton tries to guess a derivation of a given input word in a way similar to a bottom-up parser and during the computation it preserves the information about the simulated derivation by the given context-free grammar.

Theorem 2. *Any context-free language can be recognized by a nondeterministic E-automaton.*

Proof: Let L be a context-free language and let $G = (V_N, V_T, S, P)$ be a context-free grammar of L, where V_N, V_T, S and P are, respectively, the nonterminals, the terminals, the initial nonterminal and the set of rules of G. Without loss of generality we assume that the initial nonterminal S does not appear on the righthand side of any production from P and G is in Chomsky normal form, i. e.

all rules of G are of the form $A \to BC$ or $A \to a$, where A,B,C are nonterminals and a is a terminal symbol.

We will manipulate derivation trees of some strings according to the grammar G. A leaf of a derivation tree will be called a *terminal leaf* if it is marked by a terminal symbol and a *nonterminal leaf* if it is marked by a nonterminal. Define the *size* of a tree to be the number of terminal leaves of the tree. Let v be a node of a derivation tree, the subtree of this tree rooted at v we denote by T_v. Let K be a positive integer whose value will be determined later. We will use some special types of derivation subtrees (their root need not be marked by the initial nonterminal of G):

(a) Derivation tree of size at least K and at most $3K$ without nonterminal leaves.

(b) Derivation tree of size at least K and at most $2K$ with one nonterminal leaf.

(c) Derivation tree of size less than K with two nonterminal leaves.

If we have some tree T_1 of type (b) with one nonterminal leaf marked by nonterminal A and some tree T_2 of type (a) which root is marked by A, then we can construct a tree of greater size by replacing the nonterminal leaf of tree T_1 by the tree T_2. In a similar way we can construct bigger trees (not only of the forms (a) – (c)) from trees of types (a) – (c) . □

Claim 3. *For any positive integer K, every derivation subtree of any terminal string of length at least K can be constructed from subtrees of types (a) – (c) only.*

PROOF: The claim is proved by induction on size of a derivation tree.

A derivation tree of a terminal string of length at least K and at most $3K$ is obviously a tree of type (a), so the claim is true for them.

Suppose that the claim holds for trees of size less than n, where n is an integer, $n > 3K$. Let T_u be a derivation tree of a terminal string of length n. Let $u = u_1, u_2, ..., u_k$ be a path in tree T_u from node u to u_k such that subtree rooted at u_k has size at least $n - K + 1$ and subtrees T_1 and T_2 rooted at sons of the node u_k have sizes less than $n - K + 1$ (cf. Fig. 1). Such path can be found starting in the root u and choosing the path through the son whose subtree has size not less then his brother.

Thus

$$size(T_1) + size(T_2) \geq n - K + 1$$

$$size(T_1) < n - K + 1$$

$$size(T_2) < n - K + 1$$

Let the $size(T_1) \geq size(T_2)$. There are two cases:

$$size(T_1) \geq n - 2K \qquad (i)$$

$$size(T_1) < n - 2K \qquad (ii)$$

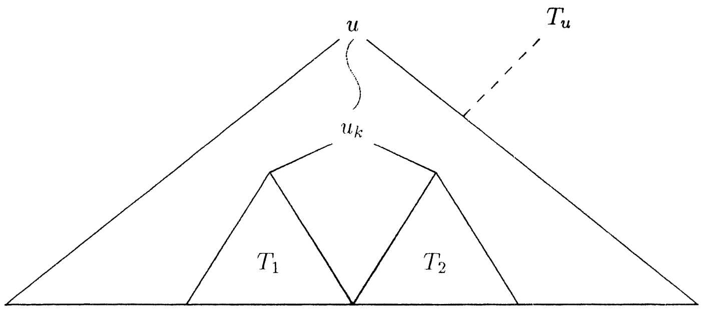

Figure 1:

In case (i) $size(T_1) > K$ because $n > 3K$. Let T' be the derivation tree obtained from T_u by replacing the subtree T_1 by a corresponding nonterminal leaf. This tree is of size at least K and less than $2K$. Thus the tree T' is of type (b). According to the induction hypothesis the claim holds for T_1 and thus it holds for tree T_u.

In case (ii) obviously

$$size(T_2) < n - 2K$$

and

$$size(T_1) > K$$

and

$$size(T_2) > K$$

then the tree obtained from T by removing subtrees T_1 and T_2 is of type (c). Again according to the induction hypothesis the claim holds for trees T_1 and T_2 and thus the claim is true for T_u.

The number of trees of types (a) – (c) for a givن grammar G is finite.

Let K be $(2|V_N| + 2)$.

Next we describe an E-automaton M which recognizes nondeterministically the language L. At the start of a computation of M the input word w is written on the list. If the word w is of length less or equal $3K$, then it is recognized directly (the number of such words is finite). Otherwise the computation of M consists of phases.

At the start of a phase there are some erased segments on the input list. An erased segment is a part of working list. It starts with a contiguous sequence of erased symbols (i. e. overwritten by @) followed by a "gap" - a sequence of nonerased original symbols of length bounded by a constant, a contiguous sequence of erased symbols and a sequence of nonerased symbols of constant length.

The main idea of the algorithm is that each erased segment corresponds to a code of an erased subtree and contains a code of the nonterminal in the root of this tree. Suppose that the nonterminals from V_N are numbered from 1 to $|V_N|$, a gap of the length i will code the i-th nonterminal, and the trailing reserve sequence

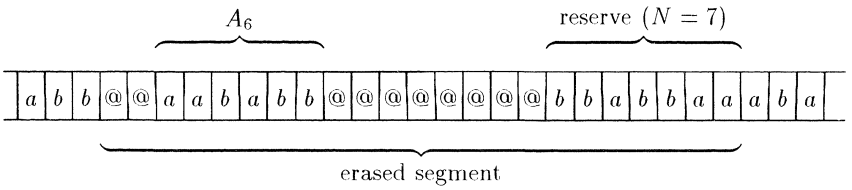

Figure 2: Erased segment which encodes sixth nonterminal A_6

will be of the length $|V_N|$. The trailing sequence of nonerased symbols is reserved for a new code when two trees are combined (cf. Fig. 2).

The *invariant of the computation* is:

> A subword consisting of original symbols represents a part of the input word, when it is the first, third, resp. the $(1 + 2i)$-th subword (for $i \geq 0$), consisting of nonerased symbols, from the left. For $i \geq 1$ the $(1 + 2i)$-th subword of original symbols (except the constant length prefix – the reserve) represents a part of the input word. The codes are represented by the $(2i)$-th original subwords from the left.

The automaton M starts each phase at the left end of the tape. M nondeterministically chooses one subtree T of the type (a) – (c). Then M scans the tape and searches substring of symbols in the leaves of T. M scans each erased segment as one unambiguously determined nonterminal. If M finds corresponding substring, then M modifies this part of the list to a new erased segment. There are three cases:

(i) T is of type (a). Then automaton M creates an erased segment on the place of the corresponding substring. This segment will code the nonterminal in the root of T. The erased tree T is of size at least K and this is enough for proper encoding.

(ii) T is of type (b). Then there is one erased segment s in the corresponding substring. Then s is preceded by a sequence of at least $K/2 = |V_N| + 1$ original symbols in this substring or s is followed by a sequence of at least $K/2 = |V_N| + 1$ original symbols in this substring.

In the former case we erase the substring so that in the part preceding s we make a code of nonterminal in the root of T, we erase the "gap" in s and we erase the sequence of nonerased symbols after the last erased symbol of s to the end of corresponding substring except the last $|V_N|$ symbols.

In the latter case we erase symbols preceding s and the "gap" in s. We modify the reserve of erased segment s to the code of the nonterminal in the root of T. The symbols following s in the corresponding substring we erase except the last $|V_N|$ symbols.

(iii) T is of type (c). Then there are two erased segments s_1 and s_2 in the corresponding substring. We erase the "gaps" in both segments and we modify the reserve of s_1 to the code of the nonterminal in the root of T. Other terminal symbols of the corresponding string except the last $|V_N|$ symbols we erase.

M finishes the computation successfully only if after some phase the list represents only one erased segment coding the initial nonterminal S of grammar G. In this case M accepts the input word.

Using the claim it is obvious that if the input word w is from $L(G)$ then there exists accepting computation of M on input w. On the other hand, M is constructed in such a way, that the mentioned invariant is preserved after each phase of a computation of M. That means, that in one phase a sequence of reductions according to the grammar G is simulated. At the end of a phase, the list represents a sentential form, from which the sentential form represented by the list at the start of this phase can be derived according to the grammar G. Thus for any accepting computation on w there is a derivation of w in the grammar G, and $w \in L(G) = L$. □

Remark. *The E-automaton was introduced in a bit different way in [1]. It was based on the notion of Turing machine with a tape bounded on the lefthand side. The real difference to our model is only in not using the left sentinel on the left margin of the tape. But in the previous construction we can mark the left margin of the list of the M by erasing the first and the third symbol of an input word. The original contents of this items can be stored in the finite control. The value of K we increase by 2 and we modify the definition of an erased segment so that the code of i-th nonterminal is a "gap" of length $i+1$. So the automaton moving from right to left can deterministically recognize the left margin of the input word as a "gap" of length one. Such an automaton need not visit the left sentinel of the list. Further the automaton must be modified so that it keeps the information which of these first three symbols are included in an erased segment. But all this can be stored in the finite control of M.*

We can see that such kind of erasing automaton is equivalent to some E-automaton. Thus erasing automata can recognize all context-free languages in a similar way as was shown in the previous construction.

Theorem 4. *The class of (deterministic) context-free languages is a proper subclass of the class of languages recognized by (deterministic) E-automata.*

PROOF: According to Theorem 2 any (deterministic) context-free language can be recognized by a (deterministic) E-automaton. On the other side, it is easy to see that the language $L_1 = \{ a^n b^n c^n \mid n > 0 \}$ can be recognized by a deterministic E-automaton (even by a d-automaton). □

The next separation theorem is an easy consequence of the Immerman and Szelepcsény well known result solving the second LBA problem. (see [3, 12]).

Theorem 5. *E-automata recognize a proper subset of the class of context sensitive languages.*

PROOF: E-automaton is a special type of linear bounded automaton, so obviously each language accepted by an E-automaton is context sensitive. On the other side, using diagonalization, we can construct a context sensitive language, which cannot be recognized by any E-automaton. The next construction follows the construction from [2].

All E-automata with some input alphabet Σ (Σ having at least two symbols) can be encoded using only symbols from Σ. Next a linear bounded automaton A with inputs from Σ^* can be constructed. The input to A is treated both as the encoding of some E-automaton E and as the input to E. A has the ability to simulate E, that is, accepts only when the E-automaton E accepts its own encoding.

Due to the positive solving of the second LBA problem the class of context sensitive languages is closed under complement. Thus we can construct a linear bounded automaton B computing complement to $L(A)$, which accepts the encoding of an E-automaton E if and only if E does not accept it. Then it is easy to show, that the language accepted by the automaton B cannot be accepted by an E-automaton. For suppose that E were such automaton. E has an encoding w in Σ^*. Suppose that E accepts w. Then A accepts w. So B does not accept w. Likewise, if E does not accept w, B does. In either case, B and E cannot accept the same language. □

Remark. *It is well known that the class of deterministic context-sensitive languages (defined by DLBA) is closed under complement. Therefore the diagonalization can be used also to prove the deterministic version of the previous theorem.*

References

[1] von Braunmühl, B., Verbeek, R., *Finite change automata*, Proceedings of the Fourth GI Conference on Theoretical Computer Science, Lecture Notes in Computer Science, Springer-Verlag **67** (1979), 91–100.

[2] Hopcroft, J. E., Ullman, J. D., *Formal languages and their relation to automata*, Addison-Wesley, Reading, Massachusetts, 1969.

[3] Immerman, N., *Nondeterministic space is closed under complement*, Proceedings of the 3rd Annual Conference Structure in Complexity Theory (June 1988), 14–17.

[4] Jančar, P., *Nondeterministic forgetting automata are less powerfull than deterministic linear bounded automata*, Acta Mathematica et Informatica Universitatis Ostraviensis **1** (1993), 67–74.

[5] Jančar, P., Mráz, F., Plátek, M., *Characterization of context-free languages by erasing automata*, in Proceedings of MFCS'92, Lecture Notes in Computer Science, Springer-Verlag **629** (August 1992), 307–314.

[6] Jančar, P., Mráz, F., Plátek, M., *Forgetting automata and the Chomsky hierarchy*, SOFSEM '92, 1992.

[7] Jančar, P., Mráz, F., Plátek, M., *A taxonomy of forgetting automata*, in Proceedings of MFCS '93, Lecture Notes in Computer Science, Springer-Verlag **711** (August 1993), 527–536.

[8] Mráz, F., Plátek, M., *Erasing automata recognize more than context-free languages*, in SOFSEM '91, Jasná pod Chopkom, Nízke Tatry, 1991.

[9] Mráz, F., Plátek, M., *A remark about forgetting automata*, in SOFSEM'93, Hrdoňov, 63–66, 1993.

[10] Plátek, M., Vogel, J., *Deterministic list automata and erasing graphs*, The Prague Bulletin of Mathematical Linguistics, Prague **45** (1986), 27–50.

[11] Rytter, W., *On the recognition of context-free languages*, in proceedings of Fifth Symposium on Computation Theory, Lecture Notes in computer Science, Springer-Verlag **208** (1985), 318–325.

[12] Szelepcsényi, R., *The method of forced enumeration for nondeterministic automata*, Acta Informatica **26(3)** (November 1988), 279–284.

Address: Department of Computer Science, Charles University
Malostranské náměstí 25, 118 00 Praha 1, Czech Republic
e-mail: `mraz@kki.ms.mff.cuni.cz, platek@kki.ms.mff.cuni.cz`