A note on the McCloskey's algorithm for deciding whether a regular language is a code (Extended Abstract) *

Luca D'Auria and Rosalba Zizza †

Variable-length codes were investigated in depth for the first time by Schützenberger (1955), by linking the theory of codes with classical noncommutative algebra. Briefly, a language X over a finite alphabet A is a code if and only if any word w in X^* admits a unique factorization in elements of X [2]. The test for codes goes back to Sardinas and Patterson (1953) and involves the definition of a set of words computed starting from X, which contain the suffixes of tentatives of two factorizations of w. If X is a regular set, the test ends and allows us to decide whether X is a code by checking if the empty word belongs to one of the above mentioned sets [1]. When X is a finite set, efficient algorithms have been designed by using different approaches (see [2]). All these algorithms run in in $O(nL)$, where n is the cardinality of X and L is the sum of the lengths of the words in X. The problem of testing whether a recognizable set is a code is a special case of a well-known problem in automata theory, namely testing whether a given rational expression is unambiguous. If X is a regular set which is not finite, no one of the previous algorithms can be applied. In [1, 2] it is shown that, if X is given by an unambiguous automaton $\mathcal{A}$, i.e., such that for each pair (p, q) of states there is no word which is the label of two different paths from p to q, a procedure can be applied. The main idea is to replace the computation on words by a computation on paths labelled by words. Thus, the uniqueness of factorizations for a code corresponds to the uniqueness of paths in the unambiguous automaton. We can determine whether a set X given by an unambiguous automaton $\mathcal{A}$ is a code, by computing $\mathcal{A}^*$, that recognizes X^*, and testing whether $\mathcal{A}^*$ is unambiguous [1, 2]. This can be done by inspecting the square automaton $S(\mathcal{A}^*)$, looking for paths of the form $(p, p) \xrightarrow{u} (r, s) \xrightarrow{v} (q, q)$, with $r \neq s$ states of $\mathcal{A}$. If such a path is found, then X is not a code. This can be tested in linear time in the number of the edges of $S(\mathcal{A}^*)$, i.e., in $O(n^2)$ where n is the number of the states of $\mathcal{A}$. However, if X is specified by means of an ambiguous finite state automaton $\mathcal{A}$, the previous technique cannot be applied. In addition, in order to make $\mathcal{A}$ unambiguous, in the worst case, it is not possible to avoid the exponential explosion of the number of the states. In [1, 2] is presented the *flower automaton* $\mathcal{A}_{\mathcal{D}}(X)$ as a universal automaton recognizing X, which is

*Partially supported by the **MIUR** Project *"Mathematical aspects and emerging applications of automata and formal languages"* (2007), by the **ESF** Project *"Automata: from Mathematics to Applications (AutoMathA)"* (2005-2010), by the 60% Project *"Proprietà strutturali e nuovi modelli di rappresentazione nella teoria dei linguaggi formali"* (University of Salerno, 2007) and by the 60% Project *"Estensioni della teoria dei linguaggi formali e loro proprietà strutturali"* (University of Salerno, 2008).

†Dipartimento di Informatica ed Applicazioni, Università di Salerno, 84085 Fisciano (SA) – ITALY. E-mail: zizza@dia.unisa.it

unambiguous. It is shown that X is a code if and only if the flower automaton $\mathcal{A}^*_{\mathcal{D}}(X)$ is unambiguous. Unfortunately, the flower automaton of a language has many states, and it can be also infinite. To overcome these difficulties, Head and Weber (1993) proposed an efficient algorithm to test whether a regular language X is a code, that involves the construction of non-deterministic finite transducer associated with X. In this paper we consider the different approach proposed in 1996 by McCloskey [5]. The advantage here is that no hypothesis is required on X, i.e., the algorithm works when X is a regular language given by a (DFA, NFA, ϵ-NFA) finite state automaton $\mathcal{A}$, even ambiguous. The core of the procedure is the definition of *restricted automaton* $\mathcal{A}_{\mathcal{R}}$, which is constructed starting from $\mathcal{A}$ in such a way $\mathcal{A}_{\mathcal{R}}$ is equivalent to $\mathcal{A}$. Briefly, a finite state automaton $\mathcal{A}$ is a restricted automaton if it is in restricted form, i.e., $\mathcal{A}$ has only one accepting state and there are no ϵ-transitions into that state and no transition on any kind out of that state. If $\mathcal{A}$ is not in restricted form, McCloskey describes how to construct the equivalent $\mathcal{A}_{\mathcal{R}}$ in restricted form, avoiding the explosion of the states. However the proof of the equivalence between $\mathcal{A}$ and $\mathcal{A}_{\mathcal{R}}$ is not given in [5]. Here we show that the proceduce for constructing $\mathcal{A}_{\mathcal{R}}$ is not complete, by providing a counterexample, and a way to recover the McCloskey's algorithm, maintaining the time complexity $O(n^2)$, n being the size of $\mathcal{A}$. In the following we suppose the reader familiar with classical notions in formal language and automata theory and here we fix only some notations [4]. Let Σ^* be the *free monoid* generated by a finite alphabet Σ and let $\Sigma^+ = \Sigma^* \setminus \epsilon$, where ϵ is the empty word. Furthermore, $|w|$ will be the length of $w \in \Sigma^*$, $|X|$ the cardinality of X. A *code* $C \subseteq \Sigma^*$ is a set of words such that any word in Σ^* has at most one factorization as a product of elements in C, i.e., for all $c_1, \ldots, c_h, c'_1, \ldots, c'_k \in C$ we have $c_1 \cdots c_h = c'_1 \cdots c'_k \Rightarrow h = k, \quad \forall i \in \{1, \ldots, h\} \quad c_i = c'_i$ [2]. We denote by DFA (resp. NFA, ϵ-NFA) a deterministic finite state automaton (resp. nondeterministic finite state state automaton, ϵ-nondeterministic finite state automaton). We recall that that DFA, NFA and ϵ-NFA accept exactly the same class of languages, i.e., regular languages. Let us present the McCloskey's algorithm given in [5] (for details regarding the correctness see [5]). The algorithm takes in input a finite state automaton $\mathcal{A}$ recognizing X and decides whether X is a code. Let n be the size of $\mathcal{A}$. The Step 2 of Algorithm 1 uses the following definition. A finite state automaton $\mathcal{A}$ is in *restricted form* if has only one accepting state and there are no ϵ-transitions into that state and no transition on any kind out of that state [5].

Algorithm 1 [5] Input: A finite state automaton $\mathcal{A}$ recognizing X.

- **STEP 1.** Check whether $\epsilon \in L(\mathcal{A})$. If is it so, X is not a code (by definition). This step can be implemented by using a Depth-First-Search visit on (the graph underlying) $\mathcal{A}$, by considering only the paths labelled by ϵ. This can be done in $O(n)$.

- **STEP 2.** Check whether $\mathcal{A}$ is in restricted form (this can be trivially done by using the definition). If it is not so, construct the restricted automaton $\mathcal{A}_{\mathcal{R}}$ equivalent to $\mathcal{A}$. This Step will be discussed in the next section.

- **STEP 3.** Construct the product automaton $\hat{\mathcal{A}}$ of $\mathcal{A}_{\mathcal{R}}$. We notice that $\hat{\mathcal{A}}$ used here is a variant of the classical product automaton and classical algorithms can be applied to construct $\hat{\mathcal{A}}$ in $O(n^2)$.

- **STEP 4.** Find a path in $\hat{\mathcal{A}}$ from the initial state to the (unique) final state, which passes through a *semi-final state*, i.e., a state $[p, q]$ in $\hat{\mathcal{A}}$ with $p \neq q$ and $p = f$ or $q = f$, f being the final state of $\mathcal{A}_{\mathcal{R}}$. If such a path exists, X is not a code; otherwise, X is a code. Also this step can be performed in time linear in the size of $\hat{\mathcal{A}}$, i.e., $O(n^2)$, as described in [5].

In (Section 4, [5]) the author describes how to make a finite state automaton in restricted form ($\mathcal{A}'$) in this way. Let S be the set of states of $\mathcal{A}$ from which accepting states are reachable via ϵ-paths, i.e., a path in which all edges are labelled by ϵ. To construct $\mathcal{A}'$, start with $\mathcal{A}$ but make all its states non-accepting. Introduce a new state, f, which will be the lone accepting state in $\mathcal{A}$. For every transition (p, a, s) in $\mathcal{A}$, where $s \in S$, $a \in \Sigma$, let (p, a, f) be a transition in $\mathcal{A}'$. Clearly, $\mathcal{A}'$ is in restricted form, $L(\mathcal{A}') = L(\mathcal{A})$, and the size of $\mathcal{A}'$ is bounded above by cn, where c is a small constant (and n is the size of $\mathcal{A}$) [5]. Let us now consider the finite state automaton $\mathcal{A}$ reported in Figure 1, recognizing $L(\mathcal{A}) = a + aba$. It is easy to see that $\mathcal{A}$ is not in restricted form and if we apply the above construction, we obtain the finite state automaton $\mathcal{A}'$ in Figure 1 (on the right), which is in restricted form, but which is not equivalent to $\mathcal{A}$, being $L(\mathcal{A}') = a$.

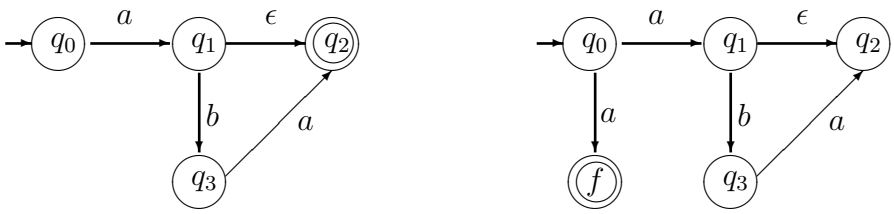
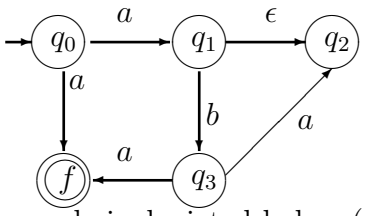
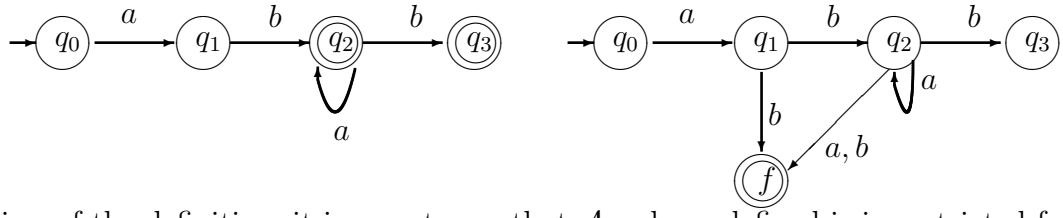


Fig.1: A finite state automaton $\mathcal{A}$ and its restricted version $\mathcal{A}'$, obtained following the original paper

We recover this construction, by giving a precise description of the construction of $\mathcal{A}_{\mathcal{R}}$. Let $\mathcal{A} = (\Sigma, Q', \delta', q_0', F')$ be an ϵ-NFA and let $\mathcal{A}_{\mathcal{R}} = (\Sigma, Q, \delta, q_0, F)$ be an ϵ-NFA defined as follows: $Q = Q' \cup \{f\}$, $f \notin Q'$, $q_0 = q_0'$, $F = \{f\}$, $\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^Q$ is defined as follows: *a*) for each $q' \in Q'$, for each $a \in \Sigma \cup \epsilon$ such that $q \in \hat{\delta}'(q', a)$, then $q \in \hat{\delta}(q', a)$. Loosely speaking, all transitions in $\mathcal{A}$ are transitions in $\mathcal{A}_{\mathcal{R}}$. *b*) for each $q' \in Q'$, for each $a \in \Sigma$ such that $q \in \delta'(q', a)$ and $q \in F'$, then $f \in \delta(q', a)$. Loosely speaking, each transition that in $\mathcal{A}$ reaches a final state in F', is transformed in a transition that reaches f in $\mathcal{A}_{\mathcal{R}}$. *c*) Let $S \subseteq Q'$ such that for each $q' \in S$, we have $\hat{\delta}'(q', \epsilon) \subseteq F'$. Let $q'' \in Q'$ such that $\delta'(q'', a) \in S$, $a \in \Sigma$. Thus in $\mathcal{A}_{\mathcal{R}}$ we add $f \in \delta(q'', a)$. This step allows us to eliminate the ϵ-paths in $\mathcal{A}$, making the paths end in f. It can be easily checked that $\mathcal{A}_{\mathcal{R}}$ is constructed in $O(n)$. The main problem of the original construction is to consider only the ϵ-paths (formalized in the item c)), whereas the definition of automaton in restricted form requires the addition of other transitions (item b)). As an example, if we apply our construction to the automaton in Figure 1 (on the left), we obtain the equivalent automaton below which is in restricted form.



Another example is depicted below (on the left the input automaton, on the right its restricted version).



In view of the definition, it is easy to see that $\mathcal{A}_{\mathcal{R}}$ above defined is in restricted form. By using the classical techniques in formal language theory, we can prove that the finite state automaton $\mathcal{A}_{\mathcal{R}}$ is equivalent to $\mathcal{A}$, i.e., $L(\mathcal{A}_{\mathcal{R}}) = L(\mathcal{A}) \setminus \epsilon$. In conclusion, having recovered the Step 2 of Algorithm 1 in time complexity $O(n)$, we have confirmed that McCloskey's algorithm allows us to decide whether a regular language recognized by a finite state automaton of size n is a code in $O(n^2)$. Our work has also concerned with the implementation of the algorithm in Java 6.0, in order to verify its feasibility in the same time complexity. The executable program (.jar file downloaded from [3]) allows to have two types of input: the diagram of a finite state automaton and a regular expression E denoting the language X, which is transformed in a NFA accepting X by using the classical theorem [4] and thanks to JFlap (http://www.cs.duke.edu/csed/jflap/). Classes have been also written to perform the translation between the JFlap data structures and our data structures.

References

[1] J. Berstel, D. Perrin, *Trends in the theory of codes*, Bulletin of the European Association Theoretical Computer Science, 29 (1986), 84 - 95.

[2] J. Berstel, D. Perrin, *Theory of Codes*, Academic Press, New York (1985).

[3] L. D'Auria, *Progetto, analisi e implementazione in Java di un algoritmo efficiente per testare l'univoca decifrabilita' di linguaggi regolari*, Tesi di Laurea Specialistica in Informatica (2010). The executable file can be downloaded at http://www.dia.unisa.it/professori/zizza/X/JRC.jar

[4] J.E. Hopcroft, R. Motwani, J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, 2nd ed., Addison-Wesley, 2001.

[5] R. McCloskey, *An $O(n^2)$ Time Algorithm for Deciding Whether a Regular Language is a Code*, Journal of Computing and Information, 2 (1996), 79 - 89, updated version downloaded at
http://www.cs.uofs.edu/∼mccloske/publications/code_alg.pdf.