MITSUBISHI ELECTRIC RESEARCH LABORATORIES
http://www.merl.com

Factorization of Finite-State Transducers

Emmanuel Roche

TR95-02 December 1995

Abstract

Finite-state transducers and finite-state automata are efficient and natural representations for a large variety of problems. We describe a new algorithm for turning a finite-state transducer into the composition of two deterministic finite-state transducers such that the combined size of the derived transducers can be exponentially smaller than other known deterministic constructions. As a consequence, this can also be used to build deterministic representations of finite-state automata smaller than the minimal finite-state automata computed by the classic determinization and minimization algorithms. We also report experimental results on large scale dictionaries and rule-based systems.

Revisions history.

1 INTRODUCTION

Finite-state transducers and finite-state automata are used in a great variety of programs such as lexical analyzers. Some of these representations can contain more than a million states in applications such as Natural Language Processing. This points out the need for efficient compaction methods.

The problem is solved in the case of deterministic finite-state automata where the minimization algorithm is well known. A minimization procedure is also available in the case of subsequential transducers [11]. For the general case of rational functions, Reutenauer and Schützenberger [12] give a way to construct a bimachine which is minimal modulo a certain equivalence relation and n the case of subsequential functions, their construction is equivalent to the minimal subsequential transducer. Building a bimachine (introduced by [14], see also [4, 2]) can also be seen as building a decomposition of a rational function f into $\alpha \circ \beta$ where α (resp. $beta$) is a right-sequential function (resp. a left-sequential function). Such a decomposition is possible for any rational function [5]. We give here an algorithm that, like in [12], builds a decomposition of any rational function into a right-sequential and a left-sequential transducer; this decomposition can be exponentially smaller than the one proposed in [12] and in the case of subsequential functions it can be exponentially smaller than the minimal subsequential transducer. In other words, there exists a family of finite-state transducers T_n such that, if $T_n = \alpha_n \circ \beta_n$ is the result of the factorization procedure presented here and if τ_n is the minimal subsequential transducer equivalent to T_n (when it exists), then $\|\alpha_n\| + \|\beta_n\| = O(log\|\tau_n\|)$.

As a curious consequence, this construction can lead to deterministic representations of finite-state automata smaller than minimal deterministic automata. A finite-state automaton can indeed be viewed as a constant rational function whose domain is the language recognized by this finite-state automaton.

In this extended abstract, we will informally describe the factorization algorithm and illustrate it by a step-by-step application on an example. The example chosen also suggests a natural proof (that will be given in the final version of this paper) that there is a family T_n such that, if $T_n = \alpha_n \circ \beta_n$ is the result of the factorization procedure presented here and if τ_n is the minimal

sequential transducer equivalent to T_n, then $\|\alpha_n\| + \|\beta_n\| = O(log\|\tau_n\|)$.[1] We will briefly say how this can be used for building finite-state representation of finite-state automata that can be smaller than the minimal deterministic finite-state automata.

We will then present some experimental results on some large scale dictionaries and rule based systems. The result show that the method is efficient and it can, in some cases, represent a big improvement upon previously known representation.

2 FACTORIZATION ALGORITHM

The algorithm can be applied to any rational function, that is to any function represented by a transducer $T = (\Sigma, Q, i, F, E)$ where Σ is the alphabet, Q is the finite set of states, $i \in Q$ is the initial state, $F \subset Q$ is the set of final states and $E \subset Q \times \Sigma \times \Sigma^* \times Q$ is the set of edges. For each edge (q, a, b, q'), also called a transition, q is called the starting state of the transition, a is called its input label, b is called its output label and q' is called its arrival state. We also define the notion of transition function by $d(q, a) = \{q' \in Q | \exists (q, a, b, q') \in E\}$ and the notion of emission function by $\delta(q, a, q') = \{b \in \Sigma^* | \exists (q, a, b, q') \in E\}$.

Let us first consider the particular case of a sequential function such as the one represented by the finite-state transducer T of Figure 1. The minimal sequential transducer τ representing the same function, that is $|\tau| = |T|$,[2] is given Figure 2. We will here show that it is possible to obtain a decomposition of T into a right-sequential transducer τ_{right} and a left-sequential transducer τ_{left} which can be smaller than the minimal sequential representation. In the case of T of Figure 1, the decomposition $T = \tau_{right} \circ \tau_{left}$ is given in Figure 3.

The algorithm is informally described on Figure 4. We shall now illustrate it by a step-by-step application on the example T of Figure 1 and show how it generates the decomposition of Figure 3. The core of this method and the reason why it improves upon previous results is introduced in the step 4 which calls the function $GRAPH\_COLORING$. This function attempts to find a minimal number of color given a graph such that no two adjacent vertices share the same color.

[1] $\|T\|$ denotes the number of states of the transducer T.

[2] If T is a finite-state transducer, $|T|$ denotes the function associated to it.

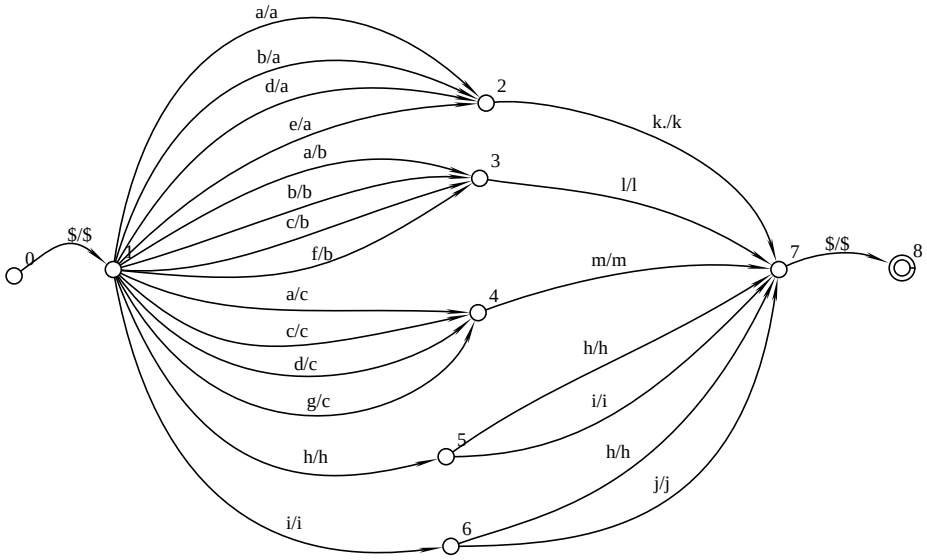

Figure 1: Transducer T.

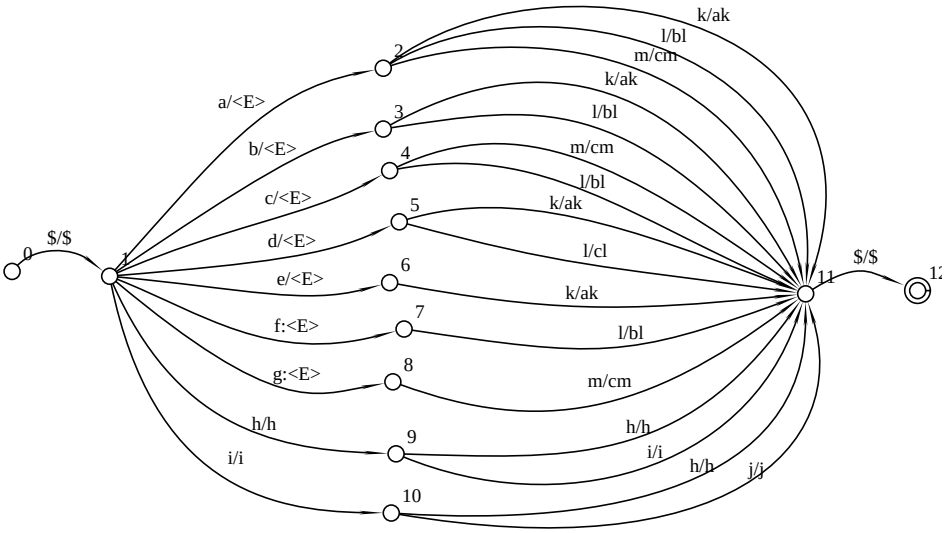

Figure 2: Subsequential transducer τ, $< E >$ stand for the empty string ϵ.

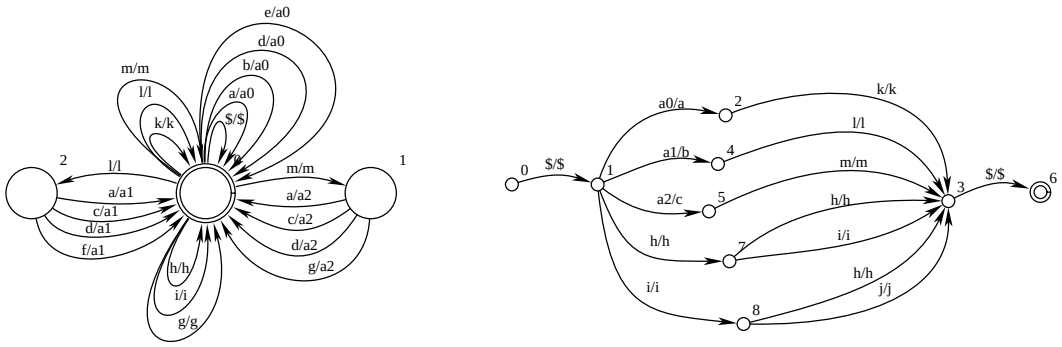

Figure 3: *left:* τ_{right}, *right:* τ_{left}

$(\tau_{right}, \tau_{left})$ = FACTORIZE(T)

1. Call A= EXTRAC_DOMAIN(T) to build the automaton A obtained from T by considering only the input symbols.
2. Call A_2=SQUARE(A) to build $A^2 = (Q_2 \subset Q \times Q, (i,i), F \times F, E_2)$.
3. Call G=BUILD_GRAPH(Q, Q_2) to generate the graph $G = (Q, E_G)$ such that $(q_1, q_2) \in E_G$ iff $(q_1, q_2) \in Q_2$.
4. Call R_G=GRAPH_COLORING(G) to build an equivalence relation R_G on Q as small as possible compatible with G, that is such that $q_1 R_G q_2 \Rightarrow (q_1, q_2)$ is not in E_G.
5. Call A_1=DET_MERGE(REV(A),R_G) to determinize the reverse of A into A_1 while identifying states that are in the same equivalence class in R_G.
6. Call T_1=ADD_STATE_CONTEXT(A_1) to transform A_1 into the transducer T_1 s.t. dom($|T_1|$) = $|A_1|$ and s.t. each output edge can be written $(q, a, (a, q), q')$.
7. Call T_2=ADJUST_LEFT(T,T_1) which builds the transducer T_2 such that $T = T_1 \circ T_2$.
8. Call R_2=MONOID(T_2) to compute the equivalence relation R_2 on $\Sigma \times Q$ such that $(a, q_1) R_2 (b, q_2)$ iff $(q, (a, q_1), b, q') \in E_2 \Leftrightarrow (q, (b, q_2), b, q') \in E_2$.
9. Call τ_{right}=REPLACE_OUTPUT(T_1,R_2) to replace each output symbol (a, q) by its equivalence class $R_2(a, q)$ and
   Call τ_{left}=REPLACE_INPUT(T_2,R_2) to replace each input symbol of T_2 by its equivalence class in R_2.

Figure 4: The factorization algorithm.

STEP 1 and 2: The fist step consists of identifying where is the left-to-right undeterminicity of the original transducer. More specifically, we look for the pairs of states (q_1, q_2) such that they both can be accessed by the same input string. For instance, for T of Figure 1, since $\$ \cdot a$ can lead to the states 2, 3 and 4, then $(2,3),(2,4)$ and $(3,4)$ are such pairs. [3]

To identify these pairs we only have to consider the input part of the transducer, that is only the input labels (left labels) on each transition. We thus first build the automaton A obtained by removing from T all the output labels (step 1). An obvious way to identify the set of pairs is to determinize the automaton A according to the classical power set construction algorithm (see [1] for instance). In fact, the fact that two states q_1 and q_2 can be reached by the same word is equivalent to the fact that there is some state set $\overline{q}$ of the powerset construction such that $q_1 \in \overline{q}$ and $q_2 \in \overline{q}$. For instance, the determinization of A is given Figure 5, it shows that the set of pairs we are looking for is the reflexive closure of $\{(2,3),(3,4),(2,4)\}$. This method however has an exponential time and space complexity and the deterministic version of A isn't used later. It is thus possible to compute the square of A (function SQUARE(A) of step 2) and get the same set of pairs. In fact, if $A = (Q, i, F, d)$ then A^2 is defined by $A^2 = (Q_2 = Q \times Q, (i,i), F \times F, d_2)$ where $d_2((q,q'),a) = d(q,a) \times d(q,b)$ and there is an equivalence between the fact that a word w reaches two different states in A and the fact that it reaches a state (q,q'), with $q \neq q'$, in A^2. Therefore, the set of pairs we look for is $Q_2 - \{(q,q)|q \in Q\}$. SQUARE(A) is partially represented in Figure 6, it leads to the same set of pairs, that is the reflexive closure of $\{(2,3),(2,4),(3,4)\}$.

STEP 3 : At this point, we just format the set of pairs as a graph whose vertices are the states of A and whose edges are the pairs (q,q') s.t. $q \neq q'$ and $(q,q') \in Q_2$ with Q_2 the states of SQUARE(Q). For our example, this leads to the graph G of Figure 6, right.

STEP 4 : The purpose of this step is to define an equivalence relation R_G on Q compatible with the graph G we just built. In other words, we look for an equivalence relation R_G such that $qR_Gq' \Rightarrow (q,q')$ is not in E_G where E_G is the set of edges of G. Moreover, we will see that in order to get a representation as small as possible, R_G should have as few equivalent classes as possible. This is exactly the definition of the extensively studied graph

[3] With the notations of [4], we look for the pairs (q_1, q_2) s.t. $i^{-1}q_1 \cap i^{-1}q_2 \neq \emptyset$.

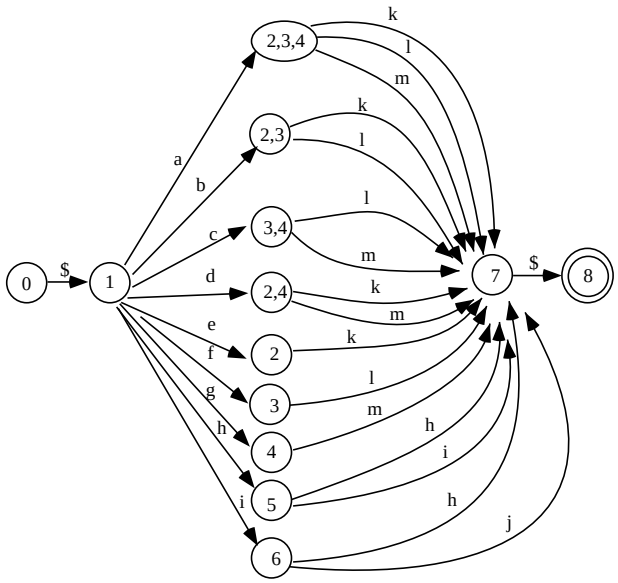

Figure 5: Determinization of A

coloring problem. Recall that given a graph $G = (Q, E_G)$, the graph coloring problem consists of finding a partition of Q into a minimum number of *color classes* $C_1, C_2, .., C_k$ where no two vertices q and q' can be in the same color class if there is an edge in E_G between them. Recall also that the graph coloring problem is NP-hard [8], thus no general optimization method for it is known. However, it is possible to use heuristics [3, 10, 7] to find a solution that, while not optimal, is not very far from the optimality (see [7] for experimental results). Here, we applied the simplest heuristic which consists in coloring the vertex in order. The vertex q_1 is assigned the color C_1 and the vertex q_i is assigned the lowest indexed color C_j that contains no vertex adjacent to q_i. If no such color exists, a new one containing only q_i is created. In our example, it created the following partition :$\{\{0, 1, 2, 5, 6, 7, 8\}, \{3\}, \{4\}\}$. We thus end up with the equivalence relation R_G s.t. $|Q/R_G| = 3$ and such that $R_G(0) = \{0, 1, 2, 5, 6, 7, 8\}$, $R_G(3) = \{3\}$ and $R_G(4) = \{4\}$.

STEP 5: The purpose of this step is to build a right-to-left deterministic automaton that encodes just enough information about the right context to separate the states of A (and therefore T) than can be reached by the same input word. In other words, we want to build A_1 such that, for each w such

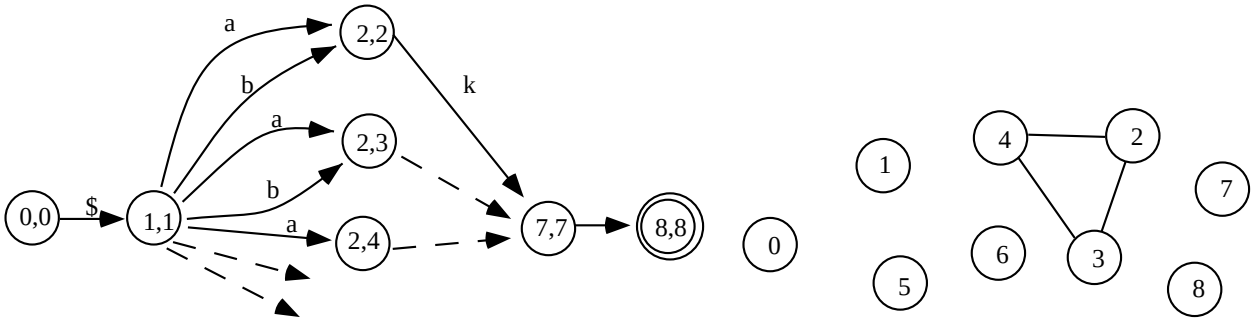

Figure 6: *left:* $A_2 = \text{SQUARE}(A)$, *right:* $G = \text{BUILD\_GRAPH}(Q, Q_2)$

that w reaches two different states q and q' of A, for each $u, v \in \Sigma^*$ s.t. $w \cdot u, w \cdot v \in |A|$ then u and v lead to two different states in A_1. We will later show how this allows to do only deterministic transitions. To compute A_1 we first have to take the reverse automaton of A, denoted $\tilde{A}$, obtained by inverting the transitions ($q' \in d_A(q, a)$ iff $q \in d_{\tilde{A}}(q', a)$) and by having the starting states $\tilde{i}$ defined by $F = \{\tilde{i}\}$ and the final states set defined by $\tilde{F} = \{i\}$ [4]. The automaton A_1 is then defined by $A_1 = (2^{Q/R_G}, R_G(\tilde{i}), R_G(\tilde{F}), d_1)$ where the transition function d_1 is defined by

$$d_1(\overline{q}, a) = R_G(\bigcup_{R(q_i) \in \overline{q}} \bigcup_{q_j \in R(q_i)} d_{\tilde{A}}(q, a))$$

In our example, this operation leads to the automaton A_1 of Figure 7, left.

STEP 6: To compute the image of an input word w in the final decomposition, we will first read the input from right to left and go through A_1. While doing that, we should remember the path followed in A_1. A way to do that [2] is to turn A_1 into a transducer that, for each transition, reemits the input and adds a reference to the state at which the transition started. For instance, if one reads the letter a while being at the state number 2, one emits the pair $(a, 2)$. Therefore, if one takes the string $\$ \cdot a \cdot m \cdot d \cdot g \cdot \$$ and applies it on A_1, one gets the output $(\$, 0) \cdot (a, 0) \cdot (m, 0) \cdot (g, 1) \cdot (\$, 0)$. The transducer T_1 that realizes this transformation is obtained from A_1 by adding on each transition (q, a, q') the output (q, a) (function ADD_STATE_CONTEXT) . For our example, T_1 is represented Figure 7, right.

[4] We can indeed assume that, without loss of generality, F only has one element.

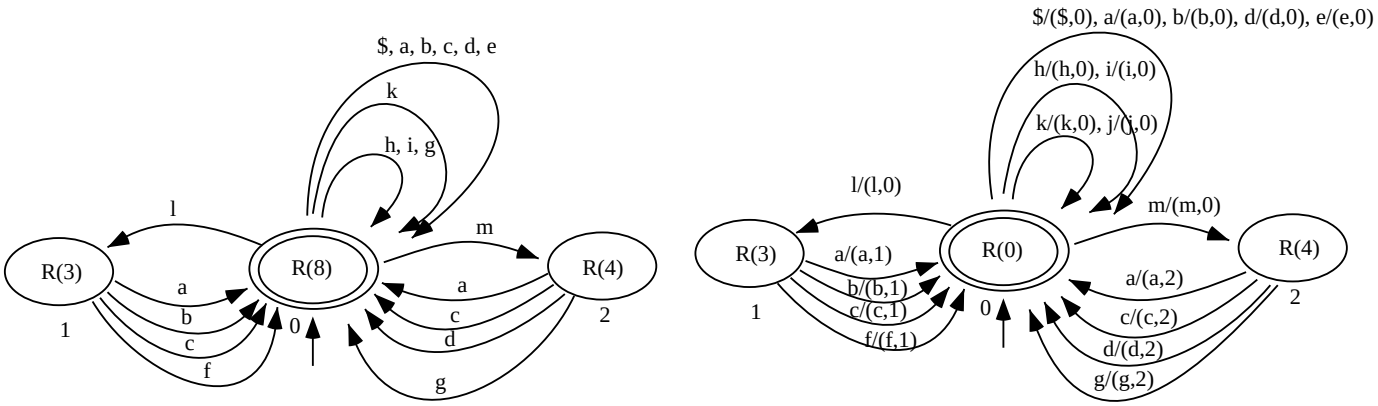

Figure 7: *left:* $A_1 = \text{DET\_MERGE}(\text{REV}(A), R_G)$ *right :* $T_1 = \text{ADD\_STATE\_CONTEXT}(A_1)$

STEP 7 : This step consists of adjusting the original transducer T such that one can first apply the transducer T_1. More precisely, we want to compute the transducer T_2 such that $|T| = |T_1| \circ |T_2|$. T_2 is obtained by looking at each transition (q, a, b, q') of T, each transition $(q_1, a, (a, q_1), q_2)$ of T_1, and if there is one word $u \cdot a \cdot v \in \text{dom}(f)$ s.t. $q \in d(i, u)$ and $q_1 \in d(i_1, v)$ then replace a by (a, q_1) in T to produce the transition $(q, (a, q_1), b, q')$. For our example, such a transducer T_2 is represented Figure 8. The exact definition of T_2 is $T_2 = (Q, i, F, E_2)$ where E_2 is defined as follows: we first define the auxiliary function g_2 that to each state of T associates the set of states of T_1 defined by

$$g_2(q) = \bigcup_{w \in \Sigma^* | d(q,w) \cap F \neq \emptyset} d_1(i_1, w)$$

E_2 is then defined by

$$E_2 = \bigcup_{(q,a,b,q') \in E} \bigcup_{q'' \in g_2(q')} \bigcup_{(q'', a, (a, q''), q) \in E_1} (q, (a, q''), b, q')$$

Obviously, $\|T_2\| = \|T\|$ and it can be proved that $|T| = |T_1| \circ |T_2|$.

STEP 8 and 9: Note that on Figure 8 the input symbols $(a, 0)$ and $(b, 0)$ behave in exactly the same way. That is, they have the same output symbols from the same starting states and to the same arrival states. Therefore, if instead of outputting two different symbols in T_1 one outputs only

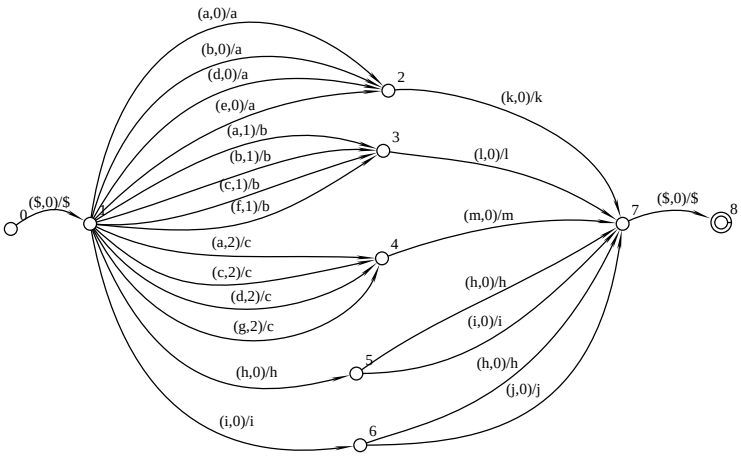

Figure 8: $T_2 = \text{ADJUST\_LEFT}(T, T_1)$

one symbol, say $(a, 0)$, it reduces the number of transitions: the transition $(1, (b, 0), a, 2)$ disappears. More formally, we can define the equivalence relation R_2 on the input symbols of T_2 by $(a, q_1)R_2(b, q_2)$ iff $(q, (a, q_1), c, q') \in E_1 \Leftrightarrow (q, (b, q_2), c, q') \in E_1$. The function MONOID returns this equivalence relation R_2. The function τ_{left}=REPLACE_INPUT(T_2,R_2) then takes each input symbol and replaces it by its equivalence class, that is

$$E_{\tau_{left}} = \{(q, R_2((a, q_1)), b, q') | (q, (a, q_1), b, q') \in E_2\}$$

In a similar way, the function τ_{right}=REPLACE_OUTPUT(T_1,R_2) takes each output symbol and replaces it by its equivalence class.

3 Experiments

In order to evaluate the factorization method we compared the size of the resulting factorization first with the minimal sequential transducer [11] when it exists and second with the bimachine factorization as given in [2]. We give the size (number of states and number of transitions) of the sequential transducer on the first line. The size of the bimachine factorization is given

on the second line and the size of the factorization we obtain with the method presented here is given on the third line.

We first took a dictionary that to each inflected word of French associates its phonetic transcription [9], called *DELAPF*. In that case, we obtain a factorization smaller that the sequential representation.

We then encoded two sets of syntactic constraints taken from two different syntactic classes from [6] as described in [13]. The finite functions that we represent take a simple sentence described by this part of the grammar, and associates to it the same simple sentence with parenthesis around the noun phrases. These functions are easy to represent by non-deterministic transducers but they are not sequential. Trying to convert such functions into a sequential transducer results in an infinite number of states. The complexity of these functions is due to the fact that the sentence structures are lexically very sensitive to the main verb. In that case, we observe that the factorization we present here is an order of magnitude smaller than the classical factorization. In that case, the factorization presented here is the only reasonable deterministic representation.

Note that the compression capabilities of this method relies on the availability of a compact non-deterministic representation. In these experiments, the input was in such a form: in the case of the phonetic dictionary it came from the shape of the phonemic rules and in the case of syntax, the natural description is highly non-deterministic.

Table 1: Results of experiments

	Delapf		SynCons-1		SynCons-2	
	nb of states	nb of trans.	nb of states	nb of trans.	nb of states	nb of trans.
Sequ. Trans.	48,224	134,726	∞	∞	∞	∞
Bima. Fact.	95,659	245,852	244,721	511,311	>1,000,000	>1,000,000.
Presented Fact.	46,881	130,952	8,783	27,511	42,224	165,991

4 Generalization to NFA

Since any finite-state automaton A can also be defined as the domain of the rational function that maps each word in $L(A)$ to a constant word, this factorization procedure can also be applied on a finite-state automaton. In fact,

note that the automaton obtained by removing the output labels from the transducer T of Figure 1 is a non-deterministic finite-state automaton (NFA) whose minimal deterministic automaton has the same number of states as the sequential transducer τ of Figure 2. Therefore, the same method leads to a deterministic decomposition smaller than the minimal deterministic automaton. Of course, looking up a word in this decomposition takes exactly (if the transducers are represented in an appropriate manner, that is, with random access to the transitions) twice the time it takes to look it up in the minimal deterministic automaton.

5 CONCLUSION

We described an algorithm that given any finite-state function, computes a factorization of this function into the composition of a right-sequential transducer with a left-sequential transducer. In the particular case of sequential functions, this factorization can be exponentially smaller than the minimal sequential transducer. Moreover, since each factor is deterministic, the space and time complexity of computing the output of an input word w is linear in term of the length of w and is independent of the size and the complexity of the function.

The method can be applied to finite-state automata in order to build deterministic representation (deterministic factorization, in fact) of finite-state automata smaller than the minimal deterministic automaton.

Experiments performed on large scale dictionaries and large scale rule-based systems demonstrates the usefulness of the approach for various aspects of language processing.

6 References

[1] Alfred; John Hopcroft Aho and Jeffrey Ulllman. *The design and analysis of computer algorithms*. Addison Wesley, 1974.

[2] Jean Berstel. *Transductions and Context-Free Languages*. Teubner, Stuttgart, 1979.

[3] D. Brelaz. New methods to color vertices of a graph. *Comm. ACM*, 22:251–256, 1970.

[4] Samuel Eilenberg. *Automata, languages, and machines*. Academic Press, New York, 1974.

[5] C. C. Elgot and G. Mezei. On relations defined by generalized finite automata. *IBM J. of Res. and Dev.*, 9:47–65, 1965.

[6] Maurice Gross. *Méthodes en syntaxe,régime des constructions complétives*. Hermann, 1975.

[7] D.S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation; part ii, graph coloring and number partitioning. *Operations Research*, 39(3):378–406, 1991.

[8] R. M. Karp. *Reducibility among combinatorial problems*, in R. E. Miller and J. W. Thatcher (eds.) 'Complexity of Computer Computations'. Plenum Press, New York, 1972.

[9] Eric Laporte. *Méthodes algorithmiques et lexicales de phonetisation de textes*. PhD thesis, Université Paris 7, 1988.

[10] F. T. Leighton. A graph coloring algorithm for large scheduling problems. *J. Res. Natl. Bur. Standards*, 84:489–506, 1979.

[11] Mehryar Mohri. Minimisation of sequential transducers. In *Proceedings of the Conference on Computational Pattern Matching 1994*, 1994.

[12] Christophe Reutenauer and Marcel-Paul Schűtzenberger. Minimizatnion of rational word functions. *SIAM J. Comput.*, 20:669–685, 1991.

[13] Emmanuel Roche. *Analyse Syntaxique Transformationelle du Français par Transducteurs et Lexique-Grammaire*. PhD thesis, Université Paris 7, January 1993.

[14] Marcel Paul Schűtzenberger. A remark on finite transducers. *Information and Control*, 4:185–187, 1961.