ELSEVIER Theoretical Computer Science 234 (2000) 177–201

Theoretical Computer Science

www.elsevier.com/locate/tcs

Minimization algorithms for sequential transducers

Mehryar Mohri

AT&T Laboratories – Research, 180 Park Avenue, Florham Park, NJ 07932, USA

Received September 1995; revised August 1996
Communicated by A. Apostolico

Abstract

We present general algorithms for minimizing sequential finite-state transducers that output strings or numbers. The algorithms are shown to be efficient since in the case of acyclic transducers and for output strings they operate in $O(S + |E| + |V| + (|E| - |V| + |F|) \cdot (|P_{\max}| + 1))$ steps, where S is the sum of the lengths of all output labels of the resulting transducer, E the set of transitions of the given transducer, V the set of its states, F the set of final states, and $P_{\max}$ one of the longest of the longest common prefixes of the output paths leaving each state of the transducer. The algorithms apply to a larger class of transducers which includes subsequential transducers. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Finite automata; Finite-state transducers; Rational power series; Semiring; Shortest-paths algorithms

1. Introduction

Finite-state automata and transducers are currently used in many applications ranging from lexical analyzers [1], language and speech processing [23], to the design of controllability systems in aircrafts [24]. The theory of automata includes now all these fields [27].

Very large finite-state transducers can be used in various domains of natural language processing [22]. In some applications, such as speech processing, the size of these machines exceeds one hundred million states. Reducing the size of these graphs without losing their recognition properties is then crucial.

This problem has been solved in the case of deterministic automata: any deterministic automaton admits an equivalent one with the minimal number of states, and classical algorithms can be used to compute that minimal automaton from a given one in an efficient way [1].

E-mail address: mohri@research.att.com (M. Mohri)

No computational algorithm was given in the case of transducers before the publication of a preliminary version of this paper [20]. We are precisely defining here such an algorithm for minimizing sequential transducers.

A characterization of minimal sequential transducers was first given by [8], later by [20, 28]. A procedure to produce a minimal sequential transducer equivalent to a given one was also indicated [9, 28]. Schützenberger [32] described a more general procedure which allows one to define from a given finite-state transducer a canonical finite-state transducer. In case the given transducer is sequential, that canonical transducer is also a minimal sequential transducer.

But none of these procedures can be used as an *algorithm* for minimizing sequential transducers since they do not describe the essential first step of construction of the intermediate transducer preceding the application of the classical automaton minimization. We define an algorithm, *quasi-determinization*, which allows one to construct that intermediate transducer [20].

This algorithm, which is the first stage of the algorithm for the minimization of transducers that we present, is independent of the notion of sequential transducers. It applies to any non-deterministic finite-state automaton (NFA). It affects the labels of the NFA to which it applies but does not increase the number of states. Its result is interesting in that, though it does not provide a deterministic automaton, it reduces the non-determinism. In fact, it reduces the non-determinism in the best way possible without modifying states or transitions of the automaton. We first define this quasi-determinization of NFAs and describe in detail the algorithm.

The algorithm bears some similarity to other classical algorithms such as the single-source shortest paths problem. We briefly compare these algorithms by introducing a semiring that helps to clarify analogies. We also extend the quasi-determinization to the case of automata in which labels are made of numbers.

We then recall the characterization of minimal sequential transducers and describe the entire algorithm allowing one to obtain minimal transducers from given sequential ones. Our algorithm applies in fact to a more general class of transducers: p-subsequential transducers. Subsequential transducers were introduced by [30]. They generalize sequential transducers by allowing a final emission at final states of sequential transducers. Choffrut [7] gave a characterization of these transducers that leads to an algorithm for determining whether a given transducer is subsequential [5]. Other related work in the characterization of subsequential transducers was done by [36]. p-subsequential transducers are more general machines in that they allow p ($p \geqslant 0$) emissions at final states. We briefly indicate how the minimization algorithm can be adapted to the case of p-subsequential transducers.

2. Quasi-determinization of NFAs

We consider here non-deterministic finite automata (NFAs). Our definition of NFAs differs however from the classical one in that labels can be strings, not necessarily elementary members of the alphabet. More precisely, we define an NFA G as a 5-tuple $(V, I, F, \Sigma, \Lambda, \delta)$ with

- V a finite set, the set of its states,
- $I \subseteq V$ the initial states,
- $F \subseteq V$ the set of final states,
- Σ a finite input alphabet,
- $\Lambda = (\lambda_i)_{i \in I}$ a vector of initial strings at initial states that need to be matched by a prefix of the input string,
- δ a state transition function which maps $V \times \Sigma^*$ to $\mathcal{N}^V$, the set of multisets of V. δ is a *finite* state transition function: the set $\{(q, \sigma) \in V \times \Sigma^*: \delta(q, \sigma) \neq \emptyset\}$ is assumed finite. We define E, the multiset of the transitions of G, by $E = \{(q, \sigma, \delta(q, \sigma))$: $(q, \sigma) \in V \times \Sigma\}$. Hence, labels of transitions can be strings and since we consider multisets, G may have several identical transitions linking two states.

Note that classical NFAs are also NFAs according to our definition. The NFAs we introduced have the same generative power as the classical ones. Indeed, they can be transformed into the classical ones by replacing each transition labeled with a string by consecutive transitions labeled with the letters of that string.

We denote by ε the empty string of Σ^* and by $x \wedge y$ the longest common prefix of two strings x and y in Σ^*. We also use the standard notation commonly used for monoids. In particular, for u and v in Σ^*, we denote by $u^{-1}(uv)$ the string v quotient of the *left division* of uv by u.

An input string $\sigma \in \Sigma^*$ is accepted by G if there exists a sequence of transitions that leads from an initial state $i \in I$ to a final state matching $\lambda_i^{-1}\sigma$. We also denote by

- G^T the transpose of G, namely the automaton obtained from G by reversing each transition,
- *Trans*$[q]$ the multiset of transitions of G leaving $q \in V$,
- *Trans*$^T[q]$ the multiset of transitions of G^T leaving $q \in V$, or equivalently the multiset of transitions of G reaching q,
- $n(t)$ the state reached by $t \in$ *Trans*$[q]$ and $l(t)$ its label in G, and also $n(t)$ the state reached by $t \in$ *Trans*$^T[q]$ and $l(t)$ its label in G^T,
- *out-degree*$[q]$ the number of transitions leaving $q \in V$,
- *in-degree*$[q]$ the number of transitions entering $q \in V$.

2.1. Definition

In the following, we consider an NFA G. We assume that the set of initial states of G is reduced to a singleton that we denote by i, $I = \{i\}$, and we denote by λ the initial string at i. Our results can be straightforwardly extended to the case of several initial states. We also assume that all states of G are coaccessible: they admit at least one path to a final state.

Let P be the function mapping V to Σ^* that associates with each state q the longest common prefix of the labels of all paths leading from q to a final state. P is well-defined since all states admit at least one path to a final state. More precisely, we define P by the following recursive definition:

$$\begin{aligned} &\forall q \in (V - F), \quad P(q) = \bigwedge_{t \in Trans[q]} l(t)\ P(n(t)), \\ &\forall q \in F, \ P(q) = \varepsilon. \end{aligned} \tag{1}$$

To simplify the following presentation, we assume that: $\lambda = P(i) = \varepsilon$. Since the definition of P is independent of λ, the general case can be treated in the same way by replacing λ by $\lambda \cdot P(i)$. Given an NFA G, we define $p(G)$, *the prefix of* G, as the non-deterministic automaton that has the same set of states and transitions as G, the same initial state and final states, and that only differs from G by the labels of its transitions in the following way: for any transition $e \in E$ with starting state q and destination state r,

$$label_{p(G)}(e) = [P(q)]^{-1} label_G(e) P(r), \tag{2}$$

where $label_{p(G)}(e)$ is the label of the transition e in $p(G)$ and $label_G(e)$ its label in G. $p(G)$ is well-defined since $P(q)$ is by definition a prefix of $label_G(e)P(r)$. It is easy to show that $p(G)$ recognizes the same language as G. $p(G)$ is obtained from G by *pushing* labels as much as possible from final states towards the initial state.

2.2. Computation

Definition (1) of function P suggests a recursive algorithm to compute $p(G)$ from G. This would suggest computing P for all states of the adjacent list of $u \in V$ before computing $P(u)$. However, in general, there does not exist a linear ordering of all states of G such that if the adjacent list of u contains v, then v appears before u in the ordering, two states can indeed belong to a cycle.

2.2.1. Acyclic case

In case G is a *dag* (directed acyclic graph) such an ordering exists. The reverse ordering of a *topological sort* of a dag meets this condition. It can be obtained in linear time $O(|V| + |E|)$, it corresponds to the increasing ordering of the finishing times in a depth-first search of G [34]. Therefore, in case G is acyclic, we can consider states according to this ordering and compute for each of them the longest common prefix corresponding to the definition (1) of P. In this way, the longest common prefix computation is performed at most once for each state of G. The computation of the automaton $p(G)$ from G can be performed in a similar way by considering the states of G in the same ordering.

2.2.2. Non-acyclic case

In case G is not acyclic, we consider strongly connected components of G. Recall that the strongly connected components, SCCs, of a directed graph are the equivalence classes of its states under the relation R defined by $q \; R \; q'$ if q' can be reached from q and q from q'.

An NFA G can be decomposed into its strongly connected components [33]. The corresponding decomposition, the *component graph* of G, is the dag G^{SCC} that has one state for each SCC of G, and a transition $(u, a, v) \in V \times \Sigma^* \times V$ if there exists a transition from the SCC of G corresponding to u to the SCC of G corresponding to v labeled with a. It can be obtained in linear time $\mathrm{O}(|E| + |V|)$.

Since the component graph of G is a dag, there exists a linear ordering of SCCs such that if the adjacent list of a state in a SCC scc_1 contains a state of another SCC scc_2, then scc_2 appears before scc_1 in the ordering: the reverse ordering of a topological sort of the dag G^{SCC}. Thus, to compute $p(G)$ from G, we can proceed as in the acyclic case except that we also need to modify the transitions of each SCC. To do so, we solve a system of equations for each SCC.

Consider a strongly connected component scc such that all the SCCs visited before scc have been consistently modified. This is necessarily true for the first SCC considered since it admits no transition leaving it. We define:

$$I[u] = \{t \in Trans[u]\colon\ n(t) \in scc\},$$

$$O[u] = \{t \in Trans[u]\colon\ n(t) \notin scc\}.$$

Then, by definition of the function P, the following system of equations ($\forall u \in scc$):

$$\begin{aligned}
&\text{if } u \in F, \quad X_u = \varepsilon,\\
&\text{if } u \notin F \text{ and } \mathrm{O}[u] = \emptyset, \quad X_u = \left(\bigwedge_{t \in I[u]} l(t) X_{n(t)}\right),\\
&\text{if } u \notin F \text{ and } \mathrm{O}[u] \neq \emptyset, \quad X_u = \left(\bigwedge_{t \in I[u]} l(t) X_{n(t)}\right) \wedge \left(\bigwedge_{t \in O[u]} l(t)\right),
\end{aligned} \tag{3}$$

has a unique solution corresponding to the longest common prefixes at each state of scc ($\forall u \in scc$, $X_u = P(u)$). We show, using the following lemmas, how the system (3) can be solved with successive changes of variables. For $u \in scc$, define π_u by

$$\pi_u \leftarrow \left(\bigwedge_{t \in Trans[u]} l(t)\right) \text{ if } u \notin F, \tag{4}$$

$$\pi_u \leftarrow \varepsilon \text{ otherwise.}$$

Let $u_0 \in scc$ be such that $\pi_{u_0} \neq \varepsilon$, we then say that u_0 is a *candidate for a change of variable*. We define a change of variable by

$$X_{u_0} \leftarrow \pi_{u_0} Y_{u_0}.$$

Denoting by $(Y_u)_{u\in scc}$ the set of unknown variables, the new system is

$$\begin{aligned}
&\text{if } u\in F, \quad Y_u=\varepsilon,\\
&\text{if } u\notin F \text{ and } \mathrm{O}[u]=\emptyset, \quad Y_u=\left(\bigwedge_{t\in I[u]} l'(t)Y_{n(t)}\right),\\
&\text{if } u\notin F \text{ and } \mathrm{O}[u]\neq\emptyset, \quad Y_u=\left(\bigwedge_{t\in I[u]} l'(t)Y_{n(t)}\right)\wedge\left(\bigwedge_{t\in O[u]} l'(t)\right),
\end{aligned} \tag{5}$$

with

$$\begin{aligned}
&\forall t\notin(Trans[u_0]\cup Trans^T[u_0]), \quad l'(t)=l(t),\\
&\forall t\in Trans[u_0]-(Trans[u_0]\cap Trans^T[u_0]), \quad l'(t)=\pi_{u_0}^{-1}l(t),\\
&\forall t\in Trans^T[u_0]-(Trans[u_0]\cap Trans^T[u_0]), \quad l'(t)=l(t)\pi_{u_0},\\
&\forall t\in(Trans[u_0]\cap Trans^T[u_0]), \quad l'(t)=\pi_{u_0}^{-1}l(t)\pi_{u_0}.
\end{aligned} \tag{6}$$

Lemma 1. *Let $u_0\in scc$ be such that $\pi_{u_0}\neq\varepsilon$, then*
(1) $(Y_u)_{u\in scc}$ *is a solution of* (5) *iff* $(X_u)_{u\in scc}$ *is a solution of* (3) *with*: $\forall u\in scc-\{u_0\}$, $X_u=Y_u$, $X_{u_0}=\pi_{u_0}Y_{u_0}$.
(2) $\sum_{u\in scc}|Y_u|<\sum_{u\in scc}|X_u|$.

Proof. The proof of 1 is straightforward using the fact that concatenating the same prefix to both sides of an equation or removing that prefix lead to equivalent equations. (2) is a direct consequence of (1). □

One can reiterate the same procedure from the system (5) by making another change of variable (using l' instead of l). This can be done as long as there exists a non-final state $u\in scc$ such that $\pi_u\neq\varepsilon$.

Lemma 2. *The system* (3) *admits only a finite number of changes of variable of the type defined above. The unique solution of the resulting system is* $(Z_u)_{u\in scc}$, *with* $\forall u\in scc$, $Z_u=\varepsilon$.

Proof. As noticed above, the system (3) admits a unique solution. Since each change of variable strictly reduces the total length of the unique solution of the new system (Lemma 1), only a finite number of changes of variable is possible (at most $\sum_{u\in scc}|X_u|$, if $(X_u)_{u\in scc}$ is the solution of (3)). No more change of variable is possible when: $\forall u\in scc$, $\pi_u=\varepsilon$. Clearly $(Z_u)_{u\in scc}$, with $\forall u\in scc$, $Z_u=\varepsilon$, is then a solution, it is unique (Lemma 1(1)). This proves the lemma. □

According to Lemma 1(1) concatenating the strings π_u involved in the changes of variable, one can reconstruct the unique solution of (3). However, we are mainly concerned with constructing the automaton $p(G)$. The successive changes of variable exactly modify the transitions of each strongly connected component of G into those of the target automaton $p(G)$. So, in fact we do not need to reconstruct the solution of the system (3).

Lemma 3. *Let S be a system obtained from* (3) *by successive changes of variable, and let $l'(t)$ be the new set of labels of the transitions $t \in scc$. Assume that the unique solution of S is $(Z_u)_{u \in scc}$, with $\forall u \in scc$, $Z_u = \varepsilon$, then $l'(t)$, ($t \in Trans[u]$, $u \in scc$), are exactly the labels of the transitions of scc in the target automaton $p(G)$.*

Proof. Let t be a transition of scc from $u \in scc$ to $n(t)$. Let (π_u^i), ($i \in [0, k_u]$), be the ordered set of changes of variable of the type $X_u \to \pi_u Y_u$ among those leading from (3) to S, and similarly $(\pi_{n(t)}^i)$, $i \in [0, k_{n(t)}]$, those involving the state $n(t)$. As seen from Eqs. (6), we have (in both cases, $u = n(t)$ or $u \neq n(t)$):

$$l'(t) = (\pi_u^{k_u})^{-1} \cdots (\pi_u^2)^{-1} (\pi_u^1)^{-1} \; l(t) \; \pi_{n(t)}^1 \pi_{n(t)}^2 \cdots \pi_{n(t)}^{k_{n(t)}}.$$

According to Lemmas 1 and 2, one can reconstruct the solution of (3) by concatenating the strings involved in the changes of variable, so

$$P(u) = \pi_u^{k_u} \cdots \pi_u^2 \pi_u^1, \tag{7}$$
$$P(n(t)) = \pi_{n(t)}^1 \pi_{n(t)}^2 \cdots \pi_{n(t)}^{k_{n(t)}}.$$

Hence:

$$\forall t \in Trans[u], \quad l'(t) = [P(u)]^{-1} l(t) P(n(t)). \tag{8}$$

This proves the lemma. □

To transform G into $p(G)$, our algorithm proceeds in the same way by solving a system of equations for each SCC considered in reverse topological order. The pseudocode of Fig. 1 describes the algorithm. The following gives the notation used:

- $V(G^{SCC})$ represents the set of states of the component graph of G. For $u \in V(G^{SCC})$, $SCC[u]$ denotes the set of states of the strongly connected component of V represented by u,
- Q a queue containing states,
- For $v \in V$,
  - · $INQ[v]$ is 1 iff v is in Q, 0 otherwise,
  - · $N[v]$ the number of ε-transitions leaving v after modification of the transitions $Trans[v]$ at any step during the algorithm. Initially, it is set to 0,
  - · $F[v]$ is 1 iff v is a final state or $N[v] = 0$ after a change of variable, 0 otherwise. Initially, it is set to 0.
- The function $LCP(G, v)$ called in the algorithm is such that it
  - · returns π, $\pi = \varepsilon$ if $v \in F$, the longest common prefix of all the transitions leaving v otherwise,
  - · replaces each of these transitions by dividing them at left by π,

```
QuasiDeterminization($G$)
1  for each $u \in V(G^{SCC})$ ▷ considered in order of increasing finishing
                                  times of a depth-first search of $G^{SCC}$
2  do for each $v \in SCC[u]$
3        do $N[v] \leftarrow INQ[v] \leftarrow F[v] \leftarrow 0$
4      $Q \leftarrow \{v\}$ ▷ $v$ arbitrarily chosen in $SCC[u]$
5      $INQ[v] \leftarrow 1$
6      while $Q \neq \emptyset$
7        do $v \leftarrow head[Q]$
8           DEQUEUE($Q$)
9           $INQ[v] \leftarrow 0$
10          $\pi \leftarrow LCP(G, v)$
11          for each $t \in Trans^T[v]$
12             do if $(\pi \neq \varepsilon)$
13                   then if  $(n(t) \in SCC[u]$  and  $N[n(t)] > 0$
                              and $l(t) = \varepsilon$ and $F[n(t)] = 0)$
14                           then $N[n(t)] \leftarrow N[n(t)] - 1$
15                        $l(t) \leftarrow l(t)\pi$
16                if $(INQ[n(t)] = 0$ and $N[n(t)] = 0$ and $F[n(t)] = 0)$
17                   then ENQUEUE($Q, n(t)$)
18                        $INQ[n(t)] = 1$
```

Fig. 1. Algorithm for the quasi-determinization of G.

- counts and stores in $N[v]$ the number of empty transitions,
- and, if $N[v] = 0$ after the computation of the longest common prefix or if v is a final state, gives $F[v]$ the value 1.

We use the queue Q to store the set of possible candidates for a change of variable. Initially, an arbitrarily chosen state v of $SCC[u]$ is enqueued in Q, and N and F set to 0 for all states.

A priori, after each change of variable, one needs to check all states $w \in SCC[u]$ to see if there still remains a candidate for a change of variable. To do so, one needs to compute π_w for all state $w \in SCC[u]$ as in the system (4). But the longest common prefix computation of the system (4), performed by the function LCP, is costly. We can limit the number of times it is performed by enqueuing a new state $w = n(t)$ in Q after modification of the transitions leaving it only if it can be a possible candidate for a change of variable, that is if w is not a final state ($F[w] = 0$) and no ε-transition leaves w ($N[w] = 0$). These are the conditions of line 16.

Also, if $N[w] = 0$ after a call of the function LCP, then the longest common prefix at w is equal to ε: $\pi_w = \varepsilon$. This equality will then hold at any time later since changes of variables only affect suffixes of the labels of the transitions leaving w. So, to avoid recomputing π_w, we set $F[w]$ to 1 in those situations (definition of the function LCP).

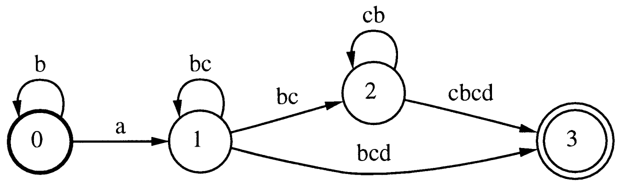

Fig. 2. Non-deterministic automaton α_1.

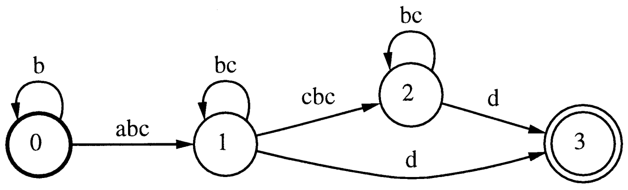

Fig. 3. NFA α_2 obtained by quasi-determinization of α_1.

The value of $N[w]$ for a non-final state w is updated every time the label of a new transition leaving w is found to be ε (lines 13 and 14). According to Lemma 3, any order for the changes of variables will lead eventually to the desired modification of G into $p(G)$ so neither the choice of the initial state, nor that of the order of the changes of variables matter here. The algorithm terminates ($Q = \emptyset$) and is correct according to Lemma 3.

Figs. 2 and 3 illustrate this algorithm. Note that the non-determinism that appears in the automaton α_1 after reading *abc* for instance does not appear in α_2 after reading the same string.

2.3. Complexity

The computation of the longest common prefix of n ($n > 1$) words requires at most $(|\pi| + 1) \cdot (n - 1)$ comparisons, where π is the result of the computation. Indeed, this operation consists of comparing the letters of the first word to those of the $(n - 1)$ others until a mismatch or an end of word occurs. The same comparisons allow one to obtain the left division by π and the number of empty transitions. In case only one transition leaves v, the computation of the longest common prefix can be assumed to be in O(1). Hence, the cost of a call of the function *LCP* for a state $v \in V - F$ is in $O((|\pi| + 1)(\mathit{out\text{-}degree}(v) - 1) + 1)$ where π is the longest common prefix of the transitions leaving v. Since π is a factor of $P(v)$ (Lemma 1, the total cost of the longest common prefix computation for a given state v is in $O((|P(v)| + 1)(\mathit{out\text{-}degree}(v) - 1) + 1)$.

Except for the first time, the longest common prefix computation at a state v is performed only if $N[v] \neq 0$. After the computation of the longest common prefix we have either $N[v] = 0$ and then v will never be enqueued again, or $N[v] \neq 0$ and π is

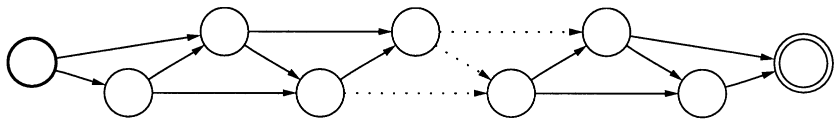

Fig. 4. Automaton G.

a new non empty factor of $P(v)$ (Lemma 1). Thus, each state v is enqueued at most $(|P(v)|+2)$ times in Q (loop of lines 6–18).

One can construct an array indicating for each $v \in V$ its corresponding SCC or state in G^{SCC}. This can be done in linear time $\mathrm{O}(|V|+|E|)$ using the SCCs of G. Using that array, the test $n(t) \in SCC[u]$ of line 13 can be performed in constant time. The cost of each iteration of the loop of lines 11–18 can then be considered as constant (O(1)). This loop is iterated *in-degree*(v) times inside the loop of lines 11–18 for each state v enqueued at line 7.

All other operations (initialization, computation of the *SCCs* and of G^{SCC}, and the definition of the reverse topological sort of G^{SCC}) can be done in $\mathrm{O}(|V|+|E|)$. Therefore, the total cost of the algorithm above is in

$$\mathrm{O}\left(|V|+|E|+\sum_{v\in V'}(\textit{out-degree}(v)-1)\cdot|P(v)|+\sum_{v\in V}\textit{in-degree}(v)\cdot|P(v)|\right), \quad (9)$$

where $V' = \{v \in V \mid \textit{out-degree}(v) > 1\}$. Let $P_{\max}$ be one of the longest of the longest common prefixes of the output paths leaving the states $v \in V$:

$$|P_{\max}| = \max_{v\in V} |P(v)|. \quad (10)$$

Then, after at most $(|P_{\max}|+2)$ steps we have $Q=\emptyset$ and the algorithm terminates. So the complexity of the algorithm is in

$$\mathrm{O}(|V|+|E|\cdot(|P_{\max}|+1)). \quad (11)$$

In case G is acyclic (see Fig. 4), each SCC is reduced to one state. Therefore, the loop of lines 7–18 is performed once for each state of G. The cost of the algorithm is then in

$$\mathrm{O}\left(|V|+|E|+\sum_{v\in V'}(\textit{out-degree}(v)-1)\cdot|P(v)|+\sum_{v\in V}\textit{in-degree}(v)\right) \quad (12)$$

hence in

$$\mathrm{O}(|V|+|E|+(|E|-(|V|-|F|))\cdot|P_{\max}|). \quad (13)$$

At worst, if all labels of the automaton are identical for instance, $|P_{\max}|$ can reach the length of the longest path from the initial state to a final state without going through cycles. The following figure represents an acyclic automaton with identical labels (all equal to an element $a \in \Sigma$), in which all states except the final one have an *out-degree* of 2. Here, we have $|P_{\max}| = |V|/2$.

At each state, the number of comparisons needed for the computation of the longest common prefix in the algorithm described above is proportional to $(d+1)$, where d is its distance to the final state. It can be proved easily that quasi-determinization runs in $O(|V|^2)$ in this case. However, in most practical cases, $|P_{\max}|$ is very small compared to $|V|$, and the algorithm can be considered to be very efficient.

An algorithm was used in the context of learning transducers to push back the output labels [25]. It corresponds to the more specific and simple case of quasi-determinization with tree-like automata.

Also, after the first publication of our algorithm [20], another variant based on the use of suffix trees was presented [6]. The complexity of that algorithm is $O(|V|+|E|+L\log|\Sigma|)$, where L is the sum of the lengths of the strings of all the transitions of the input automaton. Because of the complexity of the construction of a suffix tree, the complexity of this algorithm depends on the size of the alphabet Σ. This is not the case for the algorithm we presented.

For the sake of the comparison, we will express the complexity of each algorithm in terms of the same parameters. Let N be the maximum length of the labels of the input automaton. Note that in the worst case we would have $|\Sigma| = L = N \cdot |E|$. So the complexity C_2 of the algorithm of [6] could be expressed by: $C_2 = O(|V| + N \cdot |E| \log(N \cdot |E|))$. Since the number of transitions of a simple path to a final state is at most $|V|-1$, in the worst case the complexity C_1 of our algorithm could be expressed by $C_1 = O(|V| + N \cdot |E| \cdot |V|)$. So, for large N or $|E|$, $\log(N \cdot |E|) \ll |V|$, our algorithm has a better complexity. The algorithm of [6] has a better complexity for relatively small number of transitions and small N, $\log(N \cdot |E|) \gg |V|$. As mentioned earlier, in practice, even for large N, $P_{\max}$ is very small. For automata with bounded $|P_{\max}|$, our algorithm has a linear time behavior.

In the next section, we examine the relationships between classical shortest paths algorithms and quasi-determinization. Then we describe, in the last section, the application of quasi-determinization to the minimization of subsequential and p-subsequential transducers.

3. Related graph problems and algorithms

The quasi-determinization algorithm we just presented bears some similarity to classical shortest paths problems. We noticed that the problem of the determination of the longest common prefixes (LCP problem in short) at each state could be expressed in terms of a system of equations. Several questions arise at this point. What other problems can be expressed by the same system of equations with other operations? Could those systems be solved in the same way? Do the classical methods, possibly adopted in the case of these problems, apply here? We address some of these questions here. We define a new semiring that will help us clarify the similarity of the LCP problem with other classical ones.

3.1. Right semirings

We say that a system $(S, \oplus, \odot, \bar{0}, \bar{1})$ is a *right semiring* if

(1) $(S, \oplus, \bar{0})$ is a commutative monoid with $\bar{0}$ as the identity element for $\oplus$,

(2) $(S, \odot, \bar{1})$ is a monoid with $\bar{1}$ as the identity element for $\odot$,

(3) $\odot$ right distributes over $\oplus$:

$$\forall (a,b,c) \in S^3, \quad (a \oplus b) \odot c = (a \odot c) \oplus (b \odot c),$$

(4) $\bar{0}$ is an annihilator:

$$\forall a \in S, \quad a \odot \bar{0} = \bar{0} \odot a = \bar{0}.$$

One can define *left semirings* in a similar way. Given a left semiring $\mathscr{S} = (S, \oplus, \odot, \bar{0}, \bar{1})$, we define the dual of $\mathscr{S}$ as the right semiring $\mathscr{S}^{\perp} = (S, \oplus, \otimes, \bar{0}, \bar{1})$ where $\otimes$ is defined by

$$\forall (a,b) \in S^2, \quad a \otimes b = b \odot a.$$

$(\Sigma^* \cup \{\infty\}, \wedge, \cdot, \infty, \varepsilon)$ is a left semiring if we assign to the newly introduced element ∞ the following property:

$$\forall a \in \Sigma^* \cup \{\infty\}, \qquad \infty \wedge a = a \wedge \infty = a \tag{14}$$

which makes it the identity for the $\wedge$ operation, and

$$\forall a \in \Sigma^* \cup \{\infty\}, \qquad \infty \cdot a = a \cdot \infty = \infty \tag{15}$$

which makes it an annihilator for concatenation. Indeed, $(\Sigma^* \cup \{\infty\}, \wedge, \infty)$ and $(\Sigma^* \cup \{\infty\}, \cdot, \varepsilon)$ are then both monoids. Besides, the $\wedge$ operation is clearly commutative and concatenation left distributes over $\wedge$:

$$\forall (a,b,c) \in (\Sigma^* \cup \{\infty\})^3, \qquad a \cdot (b \wedge c) = (ab \wedge ac). \tag{16}$$

We call this semiring *the string semiring*. Notice that it is also *idempotent*:

$$\forall a \in (\Sigma^* \cup \{\infty\}), \qquad a \wedge a = a. \tag{17}$$

3.2. Quasi-determinization and single-source shortest paths problems

We define the following system of equations as *the single-source shortest paths problem* associated with the right semiring $(S, \oplus, \odot, \bar{0}, \bar{1})$ and the NFA G, with source s:

$$\begin{cases} X_s = \bar{0}, \\ \forall q \in V - \{s\}, \quad X_q = \bigoplus_{t \in Trans^T[q]} (l(t) \odot X_{n(t)}). \end{cases} \tag{18}$$

With the *tropical semiring* $(\mathscr{R}^+ \cup \{\infty\}, min, \infty, +, 0)$, the system represents the Bellman equations and defines the classical single-shortest paths problem. If we assume the set of final states reduced to a single state s, then the system (18) is similar to the one we used to define the function P in the previous section (system (1)) except that here $Trans^T[q]$ is considered instead of $Trans[q]$. Thus, in the case of a single final state, and with the dual of the string semiring, the system (18) defines exactly the LCP problem. In fact, it also defines the LCP problem in general since we can add to an NFA a new general final state to which all old final states are linked by ε-transitions without changing the problem.

This makes the similarity between the LCP problem and classical single-source shortest paths problem clearer: they are both instances of the same algebraic problem with different right semirings.

We do not address the problem of solving such a system in general here. We will describe the general solution elsewhere. Let us simply mention without proof that the LCP problem can also be algorithmically solved using a special generalization of the classical single-source shortest paths algorithms such as the Dijkstra's algorithm [12] or that of Bellman–Ford [4, 16]. The complexity of those generalized algorithms is much worse than that of the quasi-determinization algorithm. This is because in the case of the string semiring, unlike the case of the tropical semiring, it is not enough to consider the simple paths of a graph to compute single-source shortest distances. Furthermore, the cost of the computation of each longest common prefix operation in the relaxation step can reach $|P_{\max}|$.

In case the graph is acyclic, one can use a linear time algorithm to solve the single-source shortest paths algorithm. The corresponding algorithm is based on the use of a topological order of the states of the graph. The solution we previously indicated for the LCP problem in the acyclic case is very similar to that algorithm.

3.3. *Quasi-determinization of weighted directed graphs*

The analogy pointed out in the previous section also suggests the application of quasi-determinization to weighted directed graphs (directed graphs in which transitions are labeled with weights) defined with the tropical semiring $(\mathcal{R}^+ \cup \{\infty\}, min, \infty, +, 0)$. Indeed, in the same way as strings can be pushed as much as possible towards the initial state in the automaton case, in the case of weighted directed graphs weights can be pushed as much as possible towards the initial state. We can define a function P by the following equations similar to (1):

$$\begin{aligned} &\forall q \in F, P(q) = 0, \\ &\forall q \in (V - F), \quad P(q) = \min_{t \in Trans[q]} (l(t) + P(n(t))) \end{aligned} \tag{19}$$

and, assuming that $P(i) = 0$, we define the prefix $p(G)$ of G by the following equation equivalent of (2) in this context:

$$label_{p(G)}(e) = -P(q) + label_G(e) + P(r). \tag{20}$$

As previously mentioned, P can be computed by classical single-source shortest paths algorithms [3, 4, 12, 16]. The modification of the labels so that the weights be made as close as possible to the initial state (Eq. (20)) is then straightforward and can be performed in linear time $(O(|V| + |E|))$.

The case where the transitions of NFAs are labeled with numbers rather than strings is very common in speech processing for instance where the numbers can be interpreted as probabilities. The quasi-determinization is then useful as the first step of the minimization of sequential transducers that output weights [23].

In the following, we consider that application of quasi-determinization to the minimization of sequential and p-subsequential transducers.

4. Minimized sequential transducers

4.1. Definitions

A sequential transducer (ST) T is an 8-tuple $(V, i, F, \Sigma, \Delta, \lambda, \delta, \sigma)$ where

- V is the finite set of states,
- $i \in V$ the initial state,
- $F \subseteq V$ the set of final states,
- Σ and Δ finite sets corresponding respectively to the input and output alphabets of the transducer,
- $\lambda \in \Sigma^*$ a string that is concatenated to the left of the output,
- δ the state transition function which maps $V \times \Sigma$ to V,
- σ the output function which maps $V \times \Sigma$ to Δ^*.

The partial functions δ and σ can be extended to map $V \times \Sigma^*$, by the following classical recursive relations:

$$\forall s \in V,\ \forall w \in \Sigma^*,\ \forall a \in \Sigma, \delta(s, \varepsilon) = s, \qquad \delta(s, wa) = \delta(\delta(s, w), a),$$

$$\sigma(s, \varepsilon) = \varepsilon, \qquad \sigma(s, wa) = \sigma(s, w)\sigma(\delta(s, w), a). \tag{21}$$

A sequential function f is a function which can be represented by a ST. Namely, if f is represented by $T = (V, i, F, \Sigma, \Delta, \lambda, \delta, \sigma)$, then for any $w \in \Sigma^*$ such that $\delta(i, w) \in F$, $f(w) = \lambda \cdot \sigma(i, w)$. We denote by $Dom(f)$ the set of strings w for which f is defined and by $D(f)$ the set of prefixes of the strings of $Dom(f)$:

$$D(f) = \{u \in \Sigma^*: \exists w \in \Sigma^*, uw \in Dom(f)\}. \tag{22}$$

4.2. Theorems and computation

The minimization of automata can also apply to sequential transducers if one considers each pair of input and output label as a single label. However, that minimization is not enough to reduce to the minimum the size of the sequential transducers. This is because output labels of the transducer can sometimes be moved along transitions and this way make possible the merging of some states. So, in order to define a minimal sequential transducer we need an equivalence relation finer than that of automata.

For any sequential function f one can define the following relation on $D(f)$:

$$\forall (u,v) \in D(f) \times D(f), u\, R_f\, v$$
$$\Leftrightarrow\ \exists (u',v') \in \Delta^* \times \Delta^* \begin{cases} \forall w \in \Sigma^*, uw \in Dom(f) \ \Leftrightarrow\ vw \in Dom(f) \\ uw \in Dom(f) \Rightarrow u'^{-1} f(uw) = v'^{-1} f(vw). \end{cases} \tag{23}$$

It is easy to show that R_f is an equivalence relation. The following lemma shows that if there exists a ST T computing f with a number of states equal to the index of R_f, then T is a minimal transducer computing f.

Lemma 4. *If f is a sequential function, R_f has a finite number of equivalence classes. This number is less than or equal to the number of states of any ST computing f.*

Proof. Let $T = (V, i, F, \Sigma, \Delta, \lambda, \delta, \sigma)$ be a ST computing f. Choosing $u' = \lambda \cdot \sigma(i,u)$ and $v' = \lambda \cdot \sigma(i,v)$ in the above relation allows us to show easily that:

$$\forall (u,v) \in D(f) \times D(f),\ \delta(i,u) = \delta(i,v) \Rightarrow u\, R_f\, v \tag{24}$$

$\delta(i,u) = \delta(i,v)$ also defines an equivalence relation on $D(f)$. Hence, the number of equivalence classes of this relation, namely the number of accessible states of T, is greater than or equal to the number of classes of R_f. This proves the lemma. □

In the following, we define for any sequential function f a ST whose number of states is equal to the number of equivalence classes of R_f.

Theorem 1. *For any sequential function f, there exists a minimal ST computing it. Its number of states is equal to the number of equivalence classes of R_f.*

Proof. Let f be a sequential function. Let g be the function mapping Σ^* to Δ^* defined as follows:

$$\begin{aligned} &\forall u \in D(f), \quad g(u) = \bigwedge_{\substack{w \in \Sigma^* \\ uw \in Dom(f)}} f(uw), \\ &\forall u \in \Sigma^* - D(f), \quad g(u) = \varepsilon. \end{aligned} \tag{25}$$

We denote by $\bar{u}$ the equivalence class of u w.r.t. R_f and use an expression such as $u \leqslant_p v$ to indicate that u is a prefix of v.

In the following, we assume that $\lambda = g(\varepsilon) = \varepsilon$ to simplify the presentation. Since minimization does not depend on λ, the general case can be treated in the same way by outputting the string $g(\varepsilon)$ at the beginning of the recognition, that is by replacing λ with $\lambda \cdot g(\varepsilon)$.

We can define a transducer $T = (V, i, F, \Sigma, \Delta, \lambda, \delta, \sigma)$ with the following expressions:

- $V = \{\bar{u} \mid u \in D(f)\}$,
- $i = \bar{\varepsilon}$,
- $F = \{\bar{u} \mid u \in Dom(f)\}$,

- $\forall u \in D(f), \forall a \in \Sigma : ua \in D(f),\ \delta(\overline{u}, a) = \overline{ua}$,
- $\forall u \in D(f), \forall a \in \Sigma : ua \in D(f),\ \sigma(\overline{u}, a) = [g(u)]^{-1} g(ua)$.

The definitions of V and F are consistent, since, according to the previous lemma the number of equivalence classes of R_f is finite. To show that δ and σ are consistently defined we need to prove that their definitions do not depend on the choice of the element u in $\overline{u}$. This is clearly true in the case of δ, since, if u and v belong to the same class, then ua and va are also equivalent for the relation R_f. The expression defining $\sigma(\overline{u}, a)$ is well-formed because, by definition:

$$\begin{aligned} &\forall w \in \Sigma^* : uw \in Dom(f),\ g(u) \leqslant_p f(uw) \\ &\quad \Rightarrow \forall w \in \Sigma^* : u(aw) \in Dom(f),\ g(u) \leqslant_p f(u(aw)) \quad \Rightarrow g(u) \leqslant_p g(ua). \end{aligned}$$

If u and v are equivalent, according to the definition of R_f we have

$$\exists (u', v') \in \Delta^* \times \Delta^* : \forall w \in \Sigma^*,$$

$$\begin{cases} uw \in Dom(f) \Leftrightarrow vw \in Dom(f), \\ uw \in Dom(f) \Rightarrow u'^{-1} f(uw) = v'^{-1} f(vw) \end{cases}$$

and

$$\begin{cases} u(aw) \in Dom(f) \Leftrightarrow v(aw) \in Dom(f), \\ u(aw) \in Dom(f) \Rightarrow u'^{-1} f(u(aw)) = v'^{-1} f(v(aw)) \end{cases}$$

Considering the longest common prefix of each member of the above identities leads to

$$u'^{-1} g(u) = v'^{-1} g(v) \quad \text{and} \quad u'^{-1} g(ua) = v'^{-1} g(va),$$

hence

$$[g(u)]^{-1} g(ua) = [u' v'^{-1} g(v)]^{-1} u' v'^{-1} g(va) = [g(v)]^{-1} g(va).$$

Therefore, the definition of the output function is consistent. The set of strings accepted by the input automaton of T, namely by the automaton that has the same states and transitions as T and whose labels are the input labels of T, is exactly $Dom(f)$:

$$\forall u \in \Sigma^*, \quad u \in Dom(f) \Leftrightarrow \overline{u} \in F \Leftrightarrow \delta(i, u) \in F. \tag{26}$$

Also, notice that $\forall (a, b) \in D(f) \times \Sigma^*, ab \in D(f)$,

$$\sigma(\overline{\varepsilon}, ab) = \sigma(\overline{\varepsilon}, a) \sigma(\overline{a}, b) = [g(\varepsilon)]^{-1} g(a) [g(a)]^{-1} g(ab) = g(ab).$$

A recursive application of these identities leads to: $\forall u \in D(f),\ \sigma(\overline{\varepsilon}, u) = g(u)$. Now, if $u \in Dom(f)$:

$$g(u) = \bigwedge_{\substack{w \in \Sigma^* \\ uw \in Dom(f)}} f(uw) \leqslant_p f(u). \tag{27}$$

Since f is sequential, we also have

$$f(u) \leqslant_p g(u) \tag{28}$$

hence

$$u \in Dom(f) \Rightarrow \sigma(\bar{\varepsilon}, u) = g(u) = f(u). \tag{29}$$

Thus, T is a ST representing f which has the minimal number of states. This proves the theorem. □

In the following, we consider trim sequential transducers, that is sequential transducers for which every state is reachable from the initial state, and such that for any state there exists a path leading to a final state.

Considered as a $(\Sigma \times \Delta')$-automaton, where $\Delta' \subseteq \Delta^*$ is the set of its output labels, a ST $T = (V, i, F, \Sigma, \Delta, \delta, \sigma)$ can be minimized, but the corresponding algorithm [1] does not necessarily lead to a minimal transducer. We prove that once quasi-determinization has been applied to the output automaton of T, namely to the automaton which has the same states and transitions as T and whose labels are the output labels of T, the minimization of a ST using the automata minimization leads to the minimal transducer as defined above.

Given a ST $T = (V, i, F, \Sigma, \Delta, \lambda, \delta, \sigma)$, the application of quasi-determinization to the output automaton of T has no effect on the states of T, nor on its transition function. Only its output function σ is changed. We can denote by $T_2 = (V, i, F, \Sigma, \Delta, \lambda, \delta, \sigma_2)$ the resulting transducer. Let P be the function that maps V to Δ^* defined by the following recursive relations:

$$\begin{aligned} &\forall s \in V - F, \quad P(s) = \bigwedge_{a \in \Sigma} \sigma(s, a)\, P(\delta(s, a)), \\ &\forall s \in F, \quad P(s) = \varepsilon. \end{aligned} \tag{30}$$

$P(s)$ is thus the longest common prefix of the outputs of all strings accepted by the input automaton of T when read from the state s. In order to simplify this presentation, we assume that $P(i) = \varepsilon$, which is equivalent to the assumption made above: $g(\varepsilon) = \varepsilon$. According to the previous section, σ_2 is defined by

$$\forall a \in \Sigma, \forall s \in V : \delta(s, a) \neq \emptyset, \sigma_2(s, a) = [P(s)]^{-1}\ \sigma(s, a)\ P(\delta(s, a)). \tag{31}$$

So

$$\forall u \in \Sigma^* : \delta(i, u) \neq \emptyset, \sigma_2(i, u) = \sigma(i, u)\ P(\delta(i, u)).^{1} \tag{32}$$

If $\delta(i, u) \in F$, we have: $\sigma_2(i, u) = \sigma(i, u)$. Therefore, T and T_2 compute the same sequential function. Let $T_3 = (V_3, i_3, F_3, \Sigma, \Delta, \lambda, \delta_3, \sigma_3)$ be the ST obtained from T_2 using the automata minimization. We show that T_3 is a minimal sequential transducer equivalent

[1] Note that this is equivalent to: for any u in $D(f)$, $\sigma_2(i, u) = g(u)$.

to T. The idea behind this is that once the output of T has been quasi-determinized (T_2), the pair (u', v') in the definition of R_f can be taken as $(\sigma(i,u), \sigma(i,v))$.

The minimization operation can be performed by merging the equivalent states of T_2 considered as a $(\Sigma \times \Delta')$-automaton, where $\Delta' \subseteq \Delta^*$ is the set of all output labels of T_2. Two states s_1 and s_2 of T_2 are equivalent in that sense iff: [2]

$$\begin{cases} \forall w \in \Sigma^*, \quad \delta(s_1, w) \in F \iff \delta(s_2, w) \in F, \\ \delta(s_1, w) \in F \Rightarrow \forall i \in [0, |w| - 1], \\ \qquad\qquad \sigma_2(\delta(s_1, w_0 \dots w_i), w_{i+1}) = \sigma_2(\delta(s_2, w_0 \dots w_i), w_{i+1}). \end{cases}$$

Let f be the sequential function computed by T and T_2. The following lemma helps to prove that T_3 is the minimal ST computing f as defined in the previous section.

Lemma 5. *For any (u,v) in $D(f) \times D(f)$, if $\overline{u} = \overline{v}$, then the states $\delta(i,u)$ and $\delta(i,v)$ of T_2 are equivalent.*

Proof. Let (u,v) be in $D(f) \times D(f)$, $s_1 = \delta(i,u)$ and $s_2 = \delta(i,v)$. It is easy to show that $\overline{u} = \overline{v}$ implies that:

$$\forall w \in \Sigma^*, \quad \delta(s_1, w) \in F \Rightarrow \sigma_2(s_1, w) = \sigma_2(s_2, w).$$

Since for any i in $[0, n-1]$, $uR_f v$ implies $uw_0 \dots w_i R_f vw_0 \dots w_i$, for any $w = w_1 \dots w_n \in \Sigma^*$ such that $\delta(s_1, w) \in F$, we also have, $\forall i \in [0, n-1]$:

$$\sigma_2(\delta(s_1, w_0 \dots w_i), w_{i+1} \dots w_n) = \sigma_2(\delta(s_2, w_0 \dots w_i), w_{i+1} \dots w_n),$$

which is equivalent to

$$\forall i \in [0, n-1], \quad \sigma_2(\delta(s_1, w_0 \dots w_i), w_{i+1}) = \sigma_2(\delta(s_2, w_0 \dots w_i), w_{i+1}).$$

Therefore, s_1 and s_2 are equivalent states. This ends the proof of the lemma. □

Now, since T_3 is obtained by merging the equivalent states of T_2 the lemma is equivalent to

$$\forall (u,v) \in D(f) \times D(f), \quad uR_f v \Rightarrow \delta_3(i_3, u) = \delta_3(i_3, v).$$

This implies that the number of states of T_3 is less than or equal to the number of equivalence classes of R_f. Since T_3 is a ST which computes the same function as T_2, in view of Lemma 4 we prove that these two numbers are equal. T_3 is a minimal ST computing f, its states can be identified with the equivalence classes of R_f, and it is easy to show that it is exactly the minimal transducer defined in the previous section. This proves the following theorem.

[2] A string $w \in \Sigma^*$, can be decomposed by $w = w_1 \cdots w_n$ if its length $|w|$ is n. For convenience we also define: $w_0 = \varepsilon$.

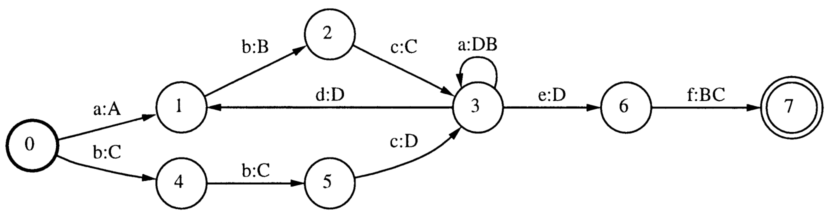

Fig. 5. Sequential transducer T.

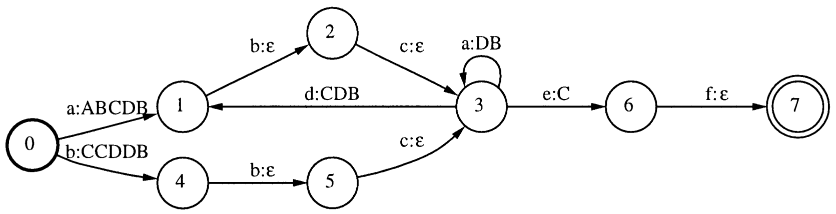

Fig. 6. Transducer T_2 obtained by quasi-determinization from T.

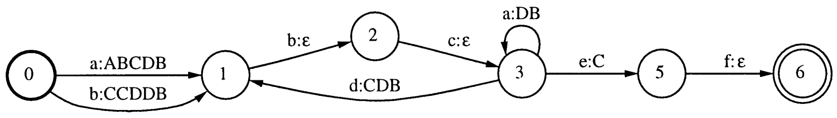

Fig. 7. Minimal sequential transducer T_3.

Theorem 2. *Given a ST T, a minimal ST computing the same function as T can be obtained by applying quasi-determinization to the output automaton of T, and the automata minimization to the resulting transducer. This minimal ST is the one defined in the previous section.*

Figs. 5–7 illustrate the minimization of sequential transducers in a particular case. Consider the ST T represented in Fig. 5. This transducer is minimal considered as an automaton. Still, it can be minimized following the process described above.

The application of quasi-determinization leads to the transducer T_2 (Fig. 6) which computes the same function. Only the output labels differ from those of T.

This ST is not minimal considered as an automaton. The application of the automata minimization leads to the reduced transducer represented in Fig. 7 which is the minimal ST as previously defined.

Transducers are often used in both directions, from inputs to outputs and vice versa. The first stage of quasi-determinization in the minimization algorithm has also an interesting effect on the reverse application of the minimal transducer. Indeed, since it reduces the ambiguities, the cost of matching a string with the output strings of the transducer is reduced.

Unlike the case of automata, in general, sequential transducers do not admit a unique minimal equivalent one. However, the minimal transducers representing the same sequential function all have the same topology and the same input labels.

Theorem 3. *Given a ST T, the minimal sequential transducers computing the same function as T only differ by the way the output labels are distributed along their paths. They have the same topology.*

Proof. The minimal sequential transducers computing the same function as T can be minimized using the algorithm described above. According to Theorem 2, the result of all these minimizations is the unique minimal transducer intrinsically defined as in Theorem 1. Since the second step of automata minimization has no effect in these minimizations, this shows that the application of quasi-determinization to the minimal transducers leads to the unique minimal transducer defined as in Theorem 1. Quasi-determinization only affect the output labels of a transducer. This proves the theorem. □

4.3. Complexity

The merging stage of the transducer minimization algorithm requires that pairs of input–output labels be identified. The input labels can be assumed to be given as integers. Using a trie with numbered leafs, one can also associate integers with output labels in linear time. Since pairs of integers can be treated as strings, the identification of pairs of labels can be done in the same way. Therefore, the whole process of identification of the input–output labels can be done in time linear in the sum of the sizes of the strings of all output labels, $O(S)$.

In the case of acyclic transducers one may use a specific minimization algorithm for automata [29] which runs in linear time. Therefore, in this case, the whole process of minimization of a ST T can be done in $O(S + |E| + |V| + (|E| - (|V| - |F|)) \cdot |P_{\max}|)$ steps, where $P_{\max}$ is one of the longest of the longest common prefixes of the output paths leaving each state of T.

In the general case, the classical automata minimization algorithm [1] runs in $O(|\Sigma| \cdot |V| \cdot \log |V|)$. It can be shown that a better implementation of the algorithm described in [1] makes it independent of the size of the alphabet. It then depends only on the in-degree of each state. Thus, a better evaluation of the running time of this algorithm is $O(|E| \cdot \log |V|)$. And, the general minimization of sequential transducers runs in $O(S + |V| + |E| \cdot (\log |V| + |P_{\max}|))$.

4.4. Minimization of sequential transducers with output weights

We described how quasi-determinization can be extended to the case of automata with output weights. Although we do not prove it here, let us mention that quasi-determinization followed by the automata minimization also leads to a minimal sequential transducer in that case [23]. More generally, quasi-determinization can be extended to the case of automata with both output string and weight. This generalized quasi-determinization can be used to minimize weighted transducers, transducers with both output string and weight, when combined with the automata minimization.

The weighted minimization algorithms can be used to reduce the size of transducers with output numbers encountered in speech recognition [23], text indexation [21], arithmetic [18], or image processing [11].

As previously mentioned, in the case of automata with output weights, the quasi-determinization stage can be performed using classical single-source shortest paths algorithms. The complexity of the whole minimization algorithm is therefore linear in the case of acyclic transducers $O(|V| + |E|)$, in $O(|E| \cdot \log |V|)$ in the case of non-acyclic graphs with no negative weights using the Dijkstra's algorithm, and in the general case of non-acyclic graphs with possibly negative numbers in $O(|V| \cdot |E|)$ [23]. Other running time complexities can be obtained in more specific cases where the size of the largest weight of the labels is small using the algorithms given by [3].

4.5. Case of p-subsequential transducers

p-Subsequential transducers generalize sequential transducers by allowing several output strings at final states (p at most). They can be represented by a 9-tuple $(V, i, F, \Sigma, \Delta, \lambda, \delta, \sigma, \Phi)$, where

- $\tau = (V, i, F, \Sigma, \Delta, \lambda, \delta, \sigma)$ is a ST, and,
- Φ is a final function mapping F to $(\Delta^*)^p$.

The result of the application of a p-subsequential transducer to a string u leading to a final state is the set of at most p distinct strings $\tau(u) \cdot \Phi(\delta(i, u))$.

The minimization algorithm for sequential transducers can be used to minimize p-subsequential transducers. To do so, we first turn a p-subsequential transducer into a sequential one, apply minimization, and then transform the result into a p-subsequential transducer.

For any $q \in F$, we denote by $(\Phi(q))_i$ the ith component of the vector $\Phi(q)$. A p-subsequential transducer $\tau = (V, i, F, \Sigma, \Delta, \lambda, \delta, \sigma, \Phi)$ can be transformed into a ST $\Psi(\tau)$ by

- Adding p new symbols ϕ_i, $1 \leqslant i \leqslant p$, to the alphabet Σ;
- Introducing a general final state f which becomes the only final state of the transducer,
- Adding transitions from each old final state q to f with input labels ϕ_i, $1 \leqslant i \leqslant p$, and output labels $(\Phi(q))_i$, $1 \leqslant i \leqslant p$.

More precisely, we perform that last step by ordering for each final state q the output labels $\Phi(q)$ in increasing lexicographic order. ϕ_1 is this way the input label of the transition with the first output label in lexicographic order. Figs. 8 and 9 illustrate that transformation at a final state q.

More formally, assuming that the final outputs of τ are lexicographically ordered at each final state, we can define $\Psi(\tau) = (V \cup \{f\}, i, F', \Sigma, \Delta \cup (\bigcup_{1 \leqslant i \leqslant p} \{\phi_i\}), \lambda, \delta', \sigma')$ by

- $f \notin V$ is a new state, and $F' = \{f\}$,

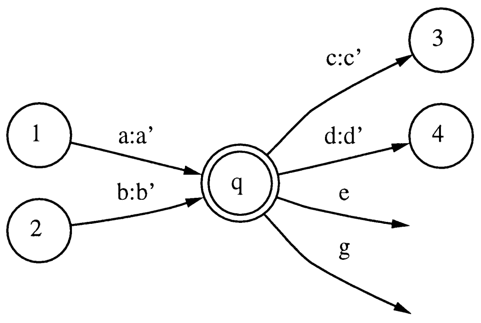

Fig. 8. p-subsequential transducer τ.

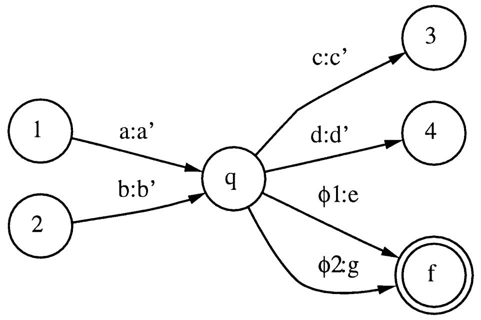

Fig. 9. Sequential transducer $\Psi(\tau)$.

- $\forall q \in V - F,\ \forall a \in \Sigma,\ \delta'(q,a) = \delta(q,a),\ \sigma'(q,a) = \sigma(q,a)$, and, $\forall q \in F,\ \forall i \in [1,p]$, $\delta'(q,\phi_i) = f,\ \sigma'(q,\phi_i) = (\Phi(q))_i$.

Let $\mathscr{T}$ be the set of sequential transducers defined over the alphabet Σ and $\Delta \cup (\bigcup_{1 \leqslant i \leqslant p} \{\phi_i\})$ that
- have a single final state f,
- only admit transitions with input labels ϕ_i, $1 \leqslant i \leqslant p$, to the state f,
- have their output labels associated with ϕ_i, $1 \leqslant i \leqslant p$, lexicographically ordered w.r.t. $i \in [1,p]$.

Similarly, define $\mathscr{T}_p$ as the set of p-subsequential transducers defined over the alphabet Σ and Δ with lexicographically sorted final outputs. Then the following lemmas give some properties of the transformation defined above.

Lemma 6. *Ψ defines a bijection mapping $\mathscr{T}_p$ to $\mathscr{T}$.*

Proof. Let Ω be the function that associates to $\tau' \in \mathscr{T}$, $\tau' = (V \cup \{f\}, i, \{f\}, \Sigma, \Delta \cup (\bigcup_{1 \leqslant i \leqslant p} \{\phi_i\}), \lambda, \delta', \sigma')$, $\tau \in \mathscr{T}_p$, $\tau = (V, i, F, \Sigma, \Delta, \lambda, \delta, \sigma, \Phi)$, with
- $F \subseteq V$ is the set of states with transitions reaching f,
- $\forall q \in V,\ \forall a \in \Sigma,\ \delta(q,a) = \delta'(q,a),\ \sigma(q,a) = \sigma'(q,a)$,

- $\forall q \in F,\ \forall i \in [1, p]$,

$$(\Phi(q))_i = \begin{cases} \sigma'(q, \phi_i) & \text{if } \sigma'(q, \phi_i) \text{ is defined,} \\ \varepsilon & \text{otherwise.} \end{cases} \tag{33}$$

It is easy to verify that the function Ω is the inverse of Ψ. □

We denote by $|\tau|$ the number of states of a transducer τ. It is straightforward from the definition of Ψ that for any $\tau \in \mathscr{T}_p$, $|\Psi(\tau)| = |\tau| + 1$. Since Ψ is a bijection (Lemma 6) we also have:

$$\forall \tau' \in \mathscr{T}, \quad |\Psi^{-1}(\tau')| = |\tau'| - 1. \tag{34}$$

We can assume without loss of generality, that the final outputs of the p-subsequential transducer to minimize have been presorted. We then have the following theorem which gives a constructive method for minimizing p-subsequential transducers.

Theorem 4. *Let τ be a p-subsequential transducer with final outputs lexicographically sorted at each final state. Let τ' be the sequential transducer obtained by minimization of $\Psi(\tau)$. Then $\Psi^{-1}(\tau')$ is a minimal p-subsequential transducer equivalent to τ.*

Proof. Quasi-determinization does not modify the lexicographic order of the transitions with input labels ϕ_i. Indeed, let $\{s_1, \ldots, s_p\}$ be a set of p strings lexicographically ordered, and denote by $\pi = \bigwedge_{1 \leqslant i \leqslant p} s_i$. Then $\{\pi^{-1}s_1, \ldots, \pi^{-1}s_n\}$ is also lexicographically ordered. Clearly the second step of automata minimization does not affect that order either. So τ' is in $\mathscr{T}$. Let $\tau_{\min} \in \mathscr{T}_p$ be a minimal p-subsequential transducer equivalent to τ. Then $\Psi(\tau_{\min})$ is a sequential transducer of $\mathscr{T}$. Thus $|\Psi(\tau_{\min})| = |\tau'|$ and using Eq. (34), $|\tau_{min}| = |\Psi^{-1}(\tau')|$. This ends the proof of the theorem. □

Note that this minimization algorithm leads to a p-subsequential transducer with sorted final outputs.

5. Conclusion

A minimization algorithm for subsequential and p-subsequential transducers with output strings was presented. This algorithm can be used in a variety of applications where large subsequential transducers are used. We gave an efficient implementation of this algorithm and used it for several applications. Our experiments in compiling very large dictionaries showed the algorithm to be very fast in practice and to be very effective in reducing the size of large transducers. As an example, using that implementation, we could compile a large French dictionary of more than 800 000 entries ($>$21 Mb), into a compact p-subsequential transducer of about 1.3 Mb in less than 20 min (including I/O's) on an HP/9000 755 [21].

When the input transducer is not deterministic, though equivalent to a p-subsequential transducer, a transducer determinization algorithm close to the classical powerset determinization can be used prior to the application of the minimization [21]. Minimization can further be used for transducers representing a (partial) rational function. Indeed, such transducers τ can be decomposed into a left-sequential transducer λ and a right-sequential transducer $\rho : \tau = \rho \circ \lambda$ [14]. It is in fact easy to construct two such sequential transducers λ and ρ, given τ [5]. Transducer minimization applies to the sequential transducers λ and ρ, this can help reduce the size of the decomposition of τ.

A minimization algorithm for subsequential transducers with output weights or both output strings and weights was also briefly described. These algorithms have been used very successfully in applications where the weights are interpreted as a measure of the probability of strings or transductions [23].

From a more theoretical point of view, the semiring we defined for the set of strings seems natural to use in several problems. It could be useful to investigate more systematically the algebraic relationships between this semiring and those used in more classical algorithms. The string semiring could give a clearer representation of some systems of equations over words which admit well-studied equivalent systems based on other semirings.

Acknowledgements

I am indebted to Maxime Crochemore and Dominique Perrin for interesting and helpful remarks. I also thank Corinna Cortes and the TCS reviewers for their comments on earlier versions of this paper.

References

[1] A.V. Aho, J.E. Hopcroft, J.D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, MA, 1974.
[2] A.V. Aho, R. Sethi, J.D. Ullman, Compilers, Principles, Techniques and Tools, Addison-Wesley, Reading, MA, 1986.
[3] R.K. Ahuja, K. Mehlhorn, J.B. Orlin, R. Tarjan, Faster algorithms for the shortest path problem, Technical Report 193, Operations Research Center, MIT, 1988.
[4] R. Bellman, On a routing problem, Quart. Appl. Math. 16 (1958)
[5] J. Berstel, Transductions and Context-Free Languages, Teubner Studienbucher, Stuttgart, 1979.
[6] D. Breslauer, The suffix tree of a tree and minimizing sequential transducers, Lecture Notes in Computer Science, vol. 1075, Springer, Berlin, 1996.
[7] C. Choffrut, Une caractérisation des fonctions séquentielles et des fonctions sous-séquentielles en tant que relations rationnelles, Theoret. Comput. Sci. 5 (1977).
[8] C. Choffrut, Contributions à l'étude de quelques familles remarquables de fonctions rationnelles, Ph.D. thesis (thèse de doctorat d'Etat), Université Paris 7, LITP, Paris, France, 1978.
[9] C. Choffrut, A generalization of Ginsburg and Rose's characterization of g-s-m mappings, Lecture Notes in Computer Science, vol. 71, Springer, Berlin, 1979.
[10] T. Cormen, C. Leiserson, R. Rivest, Introduction to Algorithms, MIT Press, Cambridge, MA, 1992.
[11] K. Culik II, J. Kari, Digital images and formal languages, in: G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, vol. 3, Springer, New York, 1997.
[12] E.W. Dijkstra, A note on two problems in connexion with graphs, Numer. Math. 1 (1959) 269–271.
[13] S. Eilenberg, Automata, Languages, and Machines, vol. A, Academic Press, New York, 1974.

[14] C.C. Elgot, J.E. Mezei, On relations defined by generalized finite automata, IBM J. Res. Develop. 9 (1965).
[15] R.W. Floyd, Algorithm 97 (SHORTEST PATH), Comm. ACM 18 (1968).
[16] L.R. Ford, D.R. Fulkerson, Flows in Network, Princeton University Press, Princeton, NJ, 1962.
[17] M.L. Fredman, R.E. Tarjan, Fibonacci heaps and their uses in improved optimalization problems, J. ACM 34 1987.
[18] C. Frougny, Synchronisation déterministe des automates à délai borné. Theoret. Comput. Sci. (1997), to appear.
[19] D.C. Kozen, The Design and Analysis of Algorithms, Springer, Berlin, 1991.
[20] M. Mohri, Minimization of sequential transducers, Lecture Notes in Computer Science, vol. 807, Springer, Berlin, 1994.
[21] M. Mohri, On some applications of finite-state automata theory to natural language processing, J. Natur. Language Eng. 2 (1996).
[22] M. Mohri, On the use of sequential transducers in natural language processing, in: Finite-State Language Processing, MIT Press, Cambridge, MA, 1997.
[23] M. Mohri, Finite-state transducers in language and speech processing, Comput. Linguistics 23 (2) (1997).
[24] A. Nerode, W. Kohn, Models for hybrid systems: automata, topologies, controllability, observability, Cornell University, Technical Report 93-28, MIT Press, Cambridge, MA, 1993.
[25] J. Oncina, P. García, E. Vidal, Learning subsequential transducers for pattern recognition and interpretation tasks, IEEE Trans. Pattern Anal. Machine Intell. 15, 1993, pp. 448–458.
[26] D. Perrin, Finite automata, in: J. Van Leuwen (Ed.), Handbook of Theoretical Computer Science, vol. B: Formal Models and Semantics, Elsevier, Amsterdam, 1990, pp. 1–57.
[27] D. Perrin, Les débuts de la théorie des automates, Technical Report LITP 93.04, Institut Blaise Pascal, Paris, France, 1993.
[28] C. Reutenauer, Subsequential functions: characterizations, minimization, examples, in: Proc. Internat. Meeting of Young Computer Scientists, Lecture Notes in Computer Science, Springer, Berlin, 1993.
[29] D. Revuz, Dictionnaires et lexiques, méthodes et algorithmes, Ph.D. thesis, Université Paris 7, Paris, France, 1991.
[30] M.P. Schützenberger, Sur une variante des fonctions séquentielles, Theoret. Comput. Sci. 4 (1977) 45–51.
[31] M.P. Schützenberger, Polynomial decomposition of rational function, in: Lecture Notes in Computer Science, vol. 386, Springer, Berlin, 1987.
[32] M.P. Schützenberger, C. Reutenauer, Minimization of rational word functions, SIAM J. Comput. 20 (4) (1991) 669–685.
[33] R.E. Tarjan, Depth first search and linear graph algorithms, SIAM J. Comput. 1 (2) (1972) 146–160.
[34] R.E. Tarjan, Finding dominators in directed graphs, SIAM J. Comput. 3 (1974) 62–89.
[35] S. Warshall, A theorem on boolean matrices, J. ACM 9 (1962) 11–12.
[36] A. Weber, R. Klemm, Economy of description for single-valued transducers, Inform. and Comput. 118 (1995) 327–340.